Contents

Working with the Result Object

When you perform server-side validations with ValidateThis the framework will return a Result object to you. You can use this Result object to provide you with a lot of information about reasons for a validation failure. This page describes many of the methods available in the Result object which return information about what happened during a server-side validation.

Getting Information about the results of a Server-Side Validation

The following methods can be called to return information about what happened during a server-side validation.

The getIsSuccess() Method

The *getIsSuccess()* method will tell you whether the validations passed or not.

Arguments

None.

Return Value

The *getIsSuccess()* method returns a boolean, which will be *true* if the validations passed and *false* if the validations failed.

Example

```
<cfset result = application.ValidateThis.validate(myObject/>
<cfif result.getIsSuccess(>
 <---  do stuff that happens when the validations pass here --->
<cfelse>
 <---  do stuff that happens when the validations fail here --->
</cfif>
```

The getFailures() Method

The *getFailures()* method will return all of the metadata available about every failure that occurred during a server-side validation. This can be useful when you want to do some additional processing using this metadata. There are other methods in the Result object, described below, which can deliver pre-processed information, ready for output.

Arguments

It accepts the following arguments:

- *locale (optional)* A locale to use to translate failure messages.

Return Value

The *getFailures()* method returns an array of structures, with each structure containing information about a single validation failure. The keys available in the structure include:

- *message* A message describing the validation failure.
- *type* The type of validation that failed (e.g., required, email, custom, etc.).
- *clientFieldname* The name of the field (from your form) that corresponds to the property that generated the failure.
- *propertyName* The name of the property that generated the failure.
- *objectType* The type of object that was validated. This corresponds to the name of your rules definition (xml) file.
- *theObject* The actual object that was validated that generated the failure.

Note that you will likely only use the first three of these keys. The last two (*objectType* and *theObject*) are generally only used when looking at failures that involved the use of the *isValidObject* validation type with a complex graph of objects.

Example

```
<cfset result = application.ValidateThis.validate(myObject)/>
<cfset allFailures = result.getFailures()/>
<cfloop array="#allFailures#" index="failure">
 <---  do something with an individual failure here --->
</cfloop>
```

The getFailureMessages() Method

The *getFailureMessages()* method will return all of the failure messages that were generated during a server-side validation. This can be useful when you simply want to output all of the messages in a single block.

Arguments

It accepts the following arguments:

- *locale (optional)* A locale to use to translate failure messages.

Return Value

The *getFailureMessages()* method returns an array of strings, with each string being a single failure message.

Example

```
<cfset result = application.ValidateThis.validate(myObject)/>
<cfset allFailureMessages = result.getFailureMessages()/>
<cfoutput>
 <ul>
  <cfloop array="#allFailureMessages#" index="failure">
   <li>#failure#</li>
  </cfloop>
 </ul>
</cfoutput>
```

This would output an unordered list of all of the failure messages that were generated during a server-side validation.

The getFailuresAsString() Method

The *getFailuresAsString()* method will return all of the failure messages that were generated during a server-side validation as a single string. This is similar to the *getFailureMessages()* method described above but instead of an array of messages you get a single string with each message delimited by a delimiter (which defaults to an html tag).

Arguments

It accepts the following arguments:

- *delim (optional)* The delimiter to use when concatenating the messages. Defaults to
.
- *locale (optional)* A locale to use to translate failure messages.

Return Value

The *getFailuresAsString()* method returns a single string, consisting of all failure messages appended using the delimiter.

Example

```
<cfset result = application.ValidateThis.validate(myObject)/>
<cfset allFailureMessages = result.getFailuresAsString()/>
<cfoutput>#allFailureMessages#</cfoutput>
```

This would output all of the failure messages, with a line break between each one.

The getFailuresByField() Method

The *getFailuresByField()* method will return all of the metadata available about every failure that occurred during a server-side validation, in a structure with one key per clientFieldname.

This can be useful when you want to do some additional processing for each field of your form (e.g., output the failures for a given field beside that field).

Arguments

It accepts the following arguments:

- *limit (optional)* The maximum number of failures to return *per field*. Defaults to an empty string which means no maximum.
- *locale (optional)* A locale to use to translate failure messages.

Return Value

The *getFailuresByField()* method returns a structure of arrays of structures, with the key to each item in the outer structure being a clientFieldname, with each item containing an array identical to the array described for the *getFailures()* method above.

Example

```
<cfset result = application.ValidateThis.validate(myObject) />
<cfset fieldFailures = result.getFailuresByField() />
<cfloop array="#fieldFailures['userName']#" index="failure">
 <---  do something with an individual failure here --->
</cfloop>
```

This would allow you to do something with all of the failures for the *userName* field.

The getFailureMessagesByField() Method

The *getFailureMessagesByField()* method will return the failure messages generated during a server-side validation, in a structure with one key per clientFieldname. This can be useful when you want to do something with the messages for each field of your form (e.g., output the messages for a given field beside that field).

Arguments

It accepts the following arguments:

- *limit (optional)* The maximum number of failures to return *per field*. Defaults to an empty string which means no maximum.
- *delim (optional)* The delimiter to use when concatenating the messages. Defaults to an empty string which generates an array. Providing any value will generate a string.
- *locale (optional)* A locale to use to translate failure messages.

Return Value

The *getFailureMessagesByField()* method returns a structure with one key per clientFieldname. Each item in the structure will either be an array of failure messages, or a string in which each failure message has been concatenated using the specified delimiter.

Example

```
<cfset result = application.ValidateThis.validate(myObject) />
<cfset fieldMessages = result.getFailureMessagesByField(delim="<br />") />
<cfoutput>#fieldMessages['userName']#</cfoutput>
```

This would output all of the failure messages generated for the *userName* field, with a line break between each one.

The getFailuresByProperty() Method

The *getFailuresByProperty()* method behaves exactly the same as the *getFailuresByField()* method, except the they keys in the structure use the *propertyName* instead of the *clientFieldname*. It is not likely that you will need to use this when generating output on an html page, but it could be useful if you are using the Result object for further processing within your model.

Please see the *getFailuresByField()* method for more information.

The getFailureMessagesByProperty() Method

The *getFailureMessagesByProperty()* method behaves exactly the same as the *getFailureMessagesByField()* method, except the they keys in the structure use the *propertyName* instead of the *clientFieldname*. It is not likely that you will need to use this when generating output on an html page, but it could be useful if you are using the Result object for further processing within your model.

Please see the *getFailureMessagesByField()* method for more information.

The getTheObject() Method

The *getTheObject()* method returns the object that was passed in for validation. This is useful if you want redirect after a failed validation and want to retrieve the populated object, so that you can prefill the form with the invalid data.

Integration with Other Frameworks

The following methods exist in the Result object to make integration with other frameworks easier.

The getFailuresForUniForm() Method

The *getFailuresForUniForm()* method returns a structure of failure messages in a format that can be sent directly to the cfUniform *form* tag. This makes integration with cfUniform a snap. It is actually equivalent to calling *getFailureMessagesByField()* and specifying
as the delimiter, but it's nice to have a shortcut.

Model-Glue Integration

The following methods exist in the Result object to provide an interface that is equivalent to the *ModelGlue.util.ValidationErrorCollection* object:

- getErrors()
- hasErrors()
- merge()

Manipulating the Result Object

During your validation processing you can manually add failures to the Result object, as well as add failures from one Result object to another Result object.

The addFailure() Method

Use the *addFailure()* method to manually add a failure to a result.

Arguments

It accepts the following arguments:

- *failure* A structure with keys that contain information about the failure. At a minimum this must include a *message* key, but should generally include keys for *propertyName* and *clientFieldname*.

Return Value

Void

Example

```
<cfset theFailure = {propertyName="userName",clientFieldname="userName",message="That's not a good user name."} />
<cfset result.addFailure(theFailure)/>
```

This would add a failure to the Result object stored in the *result* variable with a message of *That's not a good user name.* that corresponds to the property/fieldname of *userName*.

The addResult() Method

Use the *addResult()* method to add the failures from another Result object into the current Result object.

Arguments

It accepts the following arguments:

- *theResult* A Result object whose failures you wish to add to this Result object.

Return Value

Void

Example

```
<cfset result.addResult(anotherResult)/>
```

This would take all of the failures in the *anotherResult* object and add them to the Result object stored in the *result* variable.

Debugging

During development you can use the getDebugging() method of the the result object to see which rules / conditions have been evaluated by the BOValidator and whether they passed or failed. By default, debugging is disabled.

The getDebugging() Method

Use the *getDebugging()* method to view information about which rules and conditions have been evaluated.

Arguments

Return Value

The getDebugging() method returns an array of structs. For development use only, the data structure is subject to change.

Example

```
<cfset result = application.ValidateThis.validate(theObject=myObject, debuggingMode="info") />
<cfdump var="#result.getDebugging()#"/>
```

You can also enable debugging globally using the "debuggingMode" key of the ValidateThisConfig struct. Note that the global settings can be overridden per validation call as shown above.

Example

```
<cfset ValidateThisConfig = {debuggingMode="info"} />
<cfset application.ValidateThis = createObject("component","ValidateThis.ValidateThis").init(ValidateThisConfig)/>
<cfset result = application.ValidateThis.validate(theObject=myObject)/>
<cfdump var="#result.getDebugging()#"/>
```

There are three possible modes for debugging;

- none = debugging is disabled. This is the default

- info = evaluated rules / conditions and the outcome and available via the getDebugging method of the result
- strict = Same as info, with the addition of XML rules being validated for compliance against the validatethis.xsd

Please note that the output from the getDebugging() method is for help during development and is subject to change. Please don't write code which depends on it.

Deprecated Methods

The following methods have been deprecated. Please use the suggested alternatives.

The getFailuresAsStruct() Method

Use the *getFailuresByProperty()* or *getFailuresByField()* method instead.