Contents

Creating New Validation Types

ValidateThis ships with a number of built in validation types, the complete list can be found in the Validation Types Supported By ValidateThis section.

In order to add a validation type to the framework, you'll need to create two ColdFusion components (cfcs), one for the server-side validations and one for the client-side validations. We'll walk through an example of adding a simple validation type. Let's add a validatio type called *bigNumber*, which will check that a property contains a number that is al least 1,000. Of course, we could just use the built-in *min* validation type for this, but that requires us to provide a parameter (*min*) each time we want to use the rule. Let's assume that for some reason, we need to use this rule that states that the value of the property must be at least 1,000 over and over again in our model, so perhaps it makes sense to create a new validation type for it.

Creating the Server-Side Validation

Each validation type has a corresponding *Server Rule Validator* (SRV) which is responsible for performing the test to determine whether a validation passes or fails. The built-in validations have their SRVs stored in the */ValidateThis/server/* folder, so a great place to start is to find an existing SRV that is similar to the one you want to create and then use that as a basis.

As mentioned above, our validation type will not require a parameter, so let's find one that is similar. The SRV for the *numeric* validation type is a good candidate, so we'll open up the file called *ServerRuleValidator_Numeric.cfc*. Because we want our new validation type to be called *bigNumber*, we'll save a copy of this file as *ServerRuleValidator_BigNumber.cfc*. Here's what our file will look like before we change anything:

```
<cfcomponent output="false" extends="AbstractServerRuleValidator" hint="I am responsible for performing the Numeric validation.">

 <cffunction name="validate" returntype="any" access="public" output="false" hint="I perform the validation returning info in the validation object.">
  <cfargument name="valObject" type="any" required="yes" hint="The validation object created by the business object being validated." />

  <cfif shouldTest(arguments.valObject) AND NOT IsValid("Numeric",arguments.valObject.getObjectValue())>
   <cfset fail(arguments.valObject,"The #arguments.valObject.getPropertyDesc()# must be a number.") />
  </cfif>
 </cffunction>

</cfcomponent>
```

There are a number of things to note here:

- All SRVs should extend ValidateThis.server.AbstractServerRuleValidator. The SRV that we use as a template does not include the full path to AbstractServerRuleValidator.cfc because it resides in the same folder. If you plan on putting your SRV into the framework's *server* folder then this sytax is fine, but if you want to place your SRVs elsewhere (which is advisable), you'll need to specify the full path.
- You only need to provide a *validate* method, and it should accept one argument, which is the validation object that the framework passes in to it. Don't worry about how that works - you can just assume that your SRV will receive a validation object when the validate method is called.
- The code inside the validate method should perform a test and then, if the test fails, call the *fail* method, passing in the validation object and a default failure message to display. This effectively stores the validation failure in the validation object, from whence the framework will pick it up. Note that if the test passes the validate method does nothing.
- Note as well that the first part of the test that is performed above includes a call to the *shouldTest' method. This is a built-in method that* is used to support optionality in the framework. It enables validations to be optional by only performing the test if the property in question has a value. For this reason you should include a call to this method in your test, just as we see in this sample SRV.

Let's go ahead and change the above code to implement out bigNumber validation type:

```
<cfcomponent output="false" extends="ValidateThis.server.AbstractServerRuleValidator" hint="I am responsible for performing the bigNumber validation.">

 <cffunction name="validate" returntype="any" access="public" output="false" hint="I perform the validation returning info in the validation object.">
  <cfargument name="valObject" type="any" required="yes" hint="The validation object created by the business object being validated." />

  <cfif shouldTest(arguments.valObject) AND val(arguments.valObject.getObjectValue()) LT 1000>
   <cfset fail(arguments.valObject,"The #arguments.valObject.getPropertyDesc()# must be a number greater than 999.") />
  </cfif>
 </cffunction>

</cfcomponent>
```

Let's review the changes:

- We added the full path to the *extends* attribute, to allow us to store our SRV anywhere we choose.
- We updated the *hint* attribute to accurately describe the purpose of the component.
- The signature of the validate function remains unchanged.
- We kept the call to the *shouldTest* method in our if statement, and added a test for our criteria. Note that we get the value of the property in question by calling the *getObjectValue* method on the validation object which is passed into the validate method. In this case we're going to issue a validation failure if that value is less than 1000.
- Finally, we change the generated validation failure message to reflect this new test. Note how we can include the friendy name of the property in our message by using the *getPropertyDesc* method of the validation object.

We simply save this file as ServerRuleValidator_BigNumber.cfc and our server-side validation is in place.

Creating the Client-Side Validation

Each validation type has a corresponding *Client Rule Scripter* (CRS) which is responsible for generating the JavaScript required for the validation. The built-in validations have their CRSs stored in the */ValidateThis/client/{JSImplementation}/* folder, where {JSImplementation} is the name of the JavaScript implementation. We'll therefore look in */ValidateThis/client/jQuery/* to find a CRS on which to base our new validation type.

Creating these CRSs is quite a bit trickier than the SRVs because of the way the jQuery implementation code has been designed. It has been refactored to eliminate code duplication, but that makes it more difficult to follow. We'll look at creating a CRS to support this new bigNumber validation type, basing it on the 'ClientRuleScripter_Numeric.cfc *but bear in mind that you may have to do some more digging for different types of client-side validations*. Support is always available via the ValidateThis Google Group.

As before, let's open up *ClientRuleScripter_Numeric.cfc* and we'll save a copy of it as *ClientRuleScripter_BigNumber.cfc*. Here's what our file will look like before we change anything:

```
<cfcomponent output="false" extends="AbstractClientRuleScripter" hint="I am responsible for generating JS code for the numeric validation.">

 <cffunction name="getRuleDef" returntype="any" access="private" output="false" hint="I return just the rule definition which is required for the generateAddRule method.">
  <cfargument name="validation" type="any" required="yes" hint="The validation struct that describes the validation." />

  <cfreturn "number: true" />

 </cffunction>

</cfcomponent>
```

As before, there are a number of things to note here:

- All CRSs should extend ValidateThis.client.jQuery.AbstractClientRuleScripter.
- In this scenario you only need to provide a *getRuleDef* method, as all of the other logic required is in the AbstractClientRuleScripter. In other scenarios you may have to override other methods. The getRuleDef method should accept one argument, which is the validation object that the framework passes in to it.
- The code inside the getRuleDef just returns a snippet of JavaScript code which is then wrapped with some other code by other methods in the AbstractClientRuleScripter. For the ClientRuleScripter_Numeric component that method returned "number: true" as that's what the jQuery validation plugin expects for that type of validation. You'll see below that we can provide a similar rule definition for out bigNumber validation.

Let's go ahead and change the above code to implement out bigNumber validation type:

```
<cfcomponent output="false" extends="ValidateThis.client.jQuery.AbstractClientRuleScripter" hint="I am responsible for generating JS code for the bigNumber validation.">

 <cffunction name="getRuleDef" returntype="any" access="private" output="false" hint="I return just the rule definition which is required for the generateAddRule method.">
  <cfargument name="validation" type="any" required="yes" hint="The validation struct that describes the validation." />

  <cfreturn "min: 1000" />

 </cffunction>

</cfcomponent>
```

Let's review the changes:

- We added the full path to the *extends* attribute, to allow us to store our CRS anywhere we choose.
- We updated the *hint* attribute to accurately describe the purpose of the component.
- The signature of the getRuleDef function remains unchanged.
- We changed the return value of the getRuleDef method to be "min: 1000".

We simply save this file as ClientRuleScripter_BigNumber.cfc and our client-side validation is in place.

Adding the New Validation Type to the Framework

With an SRV and CRS in place, we should now be able to use our new validation type, but how will the framework know about it? The simplest way to do that is to place the files in the folders the the framework uses. So if you place your SRV in /ValidateThis/server/ and your CRS in /ValidateThis/client/jQuery/ the framework will pick them up automatically and you'll be good to go. The downside with this approach is that you've now put something in a folder that is part of the official framework, and if you were to download a new version of the framework and replace your entire installation with it you would lose your new validation type. For this reason it is possible to place your SRVs and CRSs in a folder that is outside of the framework.

You tell VT where to look for your SRVs and CRSs using two ValidateThis Config options: *extraRuleValidatorComponentPaths* and *extraClientScriptWriterComponentPaths*.

- **extraRuleValidatorComponentPaths** can contain a comma delimited list of paths to folders that contain your SRVs. When the framework loads, it will look in each of those folders for SRVs and will add any it finds to the framework. This means that you can also override built-in validation types by placing an SRV with the same name in an external folder. Any SRV that it finds will be available as a validation type.
- **extraClientScriptWriterComponentPaths** can contain a comma delimited list of paths to folders that contain your CRSs. As the name of the setting suggests, this is actually a list of paths to ClientScriptWriters, which are synonymous with JavaScript implementations. CRSs are expected to reside in a folder that corresponds to their JavaScript implementation. For the purposes of simply creating new validation types you can ignore that fact and simply put your CRSs in a folder and point to that folder.

Contributing New Validation Types to the Framework

If you have created a new validation type that you think may be useful to others, please let us know. We might just add it to the set of built-in validation types that ship with the framework.