Getting Started with VT - Redux - with or without Transfer

Posted At : April 21, 2009 11:03 PM | Posted By : Bob Silverberg
Related Categories: ColdFusion, Getting Started with VT, ValidateThis

Now that I've made ValidateThis!, my validation framework for ColdFusion objects, really easy to integrate into an existing application, I'm starting a new series about Getting Started with VT. The information covered is applicable regardless of whether you are using Transfer or not. In this first post I'm going to cover setting up VT using the new ValidateThis.cfc service object, creating a couple of validation rules, and implementing server-side validations for those rules. This is kind of a redux of the first few posts in my getting Started with VT and Transfer series, so I will be covering some familiar ground.

OK, let's get started.

Step 1 - Download the Framework

Download the latest version of VT from **validatethis.riaforge.org**.

Step 2 - Install the Framework

Unzip the contents of the /ValidateThis folder in the zip file into a folder on your ColdFusion server. To make the framework accessible you can either:

1. Place the /ValidateThis folder directly under the webroot of your application, or
2. Place the /ValidateThis folder anywhere and then create a ColdFusion mapping pointing to it via the CF Administrator or in your Application.cfc

Step 3 - Define Validation Rules for Your Object(s)

For this example we're going to add two validation rules for the UserName property of a User object. There are two ways of defining validation rules to the framework:

1. Via an xml file.
2. Via ColdFusion code, using the addRule() method.

I prefer to create xml files, so that's what we'll be doing here. I'll create a file called user.user.xml (the filename matches the Transfer class name of the object), and include in it the following code:

```
<?xml version="1.0" encoding="UTF-8"?>

<validateThis xsi:noNamespaceSchemaLocation="validateThis.xsd"

 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <objectProperties>

  <property name="UserName" desc="Email Address">

   <rule type="required" />

   <rule type="email"

    failureMessage="You call that an Email Address?" />

  </property>

 </objectProperties>

</validateThis>
```

I don't want to spend a lot of time in this post describing the format of the ValidateThis! xml schema. I described it at length in a couple of **previous posts**. In a nutshell, what we've done above is to declare that our Business Object will have a *UserName* property, and that that property has two validation rules associated with it:

1. It is required.
2. It must be a valid email address.

In addition to that we've declared that the "friendly name" of the property (to be used in validation failure messages), is *Email Address* and that the custom message that we'd like displayed when the email validation fails is "You call that an Email Address?".

Note that it is not necessary to use the Transfer class name of the business object as the name of your xml file - you can call it anything you like. When I use the more complex Business Object integration method it is important that the names match, so I try to be consistent.

Step 4 - Specify Default Values for the Framework

Create a struct that contains keys which tell ValidateThis! how to behave. At a minimum you'll probably need to tell VT where to find your xml files, so your struct may look something like this:

```
<cfset ValidateThisConfig = {definitionPath="/model/VT/"} />
```

There are a number of other keys that can be specified in the ValidateThisConfig struct, but for most of them you can accept the defaults. They keys available along with a description of what they do and their default values can be found in the **VT 0.7 Release Notes**.

Step 5 - Instantiate the ValidateThis.cfc Service Object

Create an instance of the ValidateThis.cfc service object, like so:

```
<cfset application.ValidateThis =

 createObject("component","ValidateThis.ValidateThis").init(

  ValidateThisConfig) />
```

You now have access to all of the framework's capabilities via the application.ValidateThis object.

Step 6 - Performing Server-Side Validations

To perform server-side validations on your Business Object you simply call the validate() method of the application.ValidateThis object, passing in the object type

and the object itself. The object type is the same as the filename that you used in Step 3 above, without the .xml extension. So in this example it would be "user.user". So, assuming that we have a populated object called objUser, we'd do the following:

```
<cfset Result =

 application.ValidateThis.validate(

  objectType="user.user",

  theObject=objUser) />
```

This will return a Result object to us, upon which we can invoke a number of methods including:

- getIsSuccess() - which returns *true* if all of the validations passed
- getFailures() - which returns all validation failures as an array of structs, with each struct including metadata about the failure
- getFailuresAsStruct() - which is a helper method that returns a struct with one key per property, with each key containing all of the validation failure messages

And that's it. With those six simple steps you are up and running and performing server-side validations with VT.

In the next post I'll cover how to enable client-side validations, and I'll follow that up with some more examples of different types of validations.

As always, if you have any questions or interest in following the progress of VT, I invite you to join the **ValidateThis! Google Group**.