ValidateThis! - An Object Oriented Approach to Validations

Posted At : October 6, 2008 11:41 AM | Posted By : Bob Silverberg
Related Categories: OO Design, ColdFusion, Transfer, ValidateThis

I have written in the past about the approach I take to building an object oriented model layer for my ColdFusion applications. For the most part I was happy with the way I designed things and the code that I wrote. There was one area, however, that I was never really that pleased with; validations.

So I spent the past several weeks working on a whole new approach to doing validations, and I am now ready to share it with anyone who is interested. Because I intend for this to become a standard tool in my development toolset, I am going to refer to it as a Validation Framework. I'm not sure if that is an appropriate use of the term, but it's my code and my blog, so that's what I'm going to call it.

I had the following goals for my validation framework:

Flexible Validations

It should be possible to create an unlimited number of validation types, and any validation type imaginable should be possible. It should be possible to create new validation types without having to touch any of the existing framework code. Examples of validation types are:

- Required - which includes three variations:
    - Simple - this field is always required
    - Dependent - this field is required if field x is supplied
    - Conditional - this field is required if condition x is true
- EqualTo - the value of this field must be equal to the value of field x
- Min - the value of this field must be at least x
- MinLength - the length of this field must be at least x
- Range - the value of this field must be between x and y
- RangeLength - the length of this field must be between x and y
- Date - the value of this field must be a date
- Email - the value of this field must be an Email Address
- Custom - allows for a method to be defined inside the Business Object that returns either true or false

It should also be possible to create an unlimited number of client-side validation implementations, and a new implementation can be created without having to touch any of the existing framework code. An implementation is a way of converting the business rules into JavaScript code. Examples of implementations are:

- jQuery Validation Plugin
- qForms
- Any homegrown JS validation scheme

Code Generation

One should be able to define the validation rules as business rules, and the framework will generate all server-side and client-side validation code automatically. Examples of validation rules are:

- UserName is required
- UserName must be between 5 and 10 characters long
- UserName must be unique
- Password must be equal to VerifyPassword
- If ShippingType is "Express", a ShippingMethod must be selected

The framework should be able to generate generic, but specific, validation failure messages, any of which can be overridden by an application developer. Examples of generic failure messages corresponding to the example rules above are:

- The User Name is required.
- The User Name must be between 5 and 10 characters long.
- The User Name must be unique.
- The Password must be the same as the Verify Password.
- The Shipping Method is required when selecting a Shipping Type of "Express".

Flexible Feedback

The framework should return flexible metadata back to the calling application which will allow for customization of how the validation failures will appear to the user. This metadata will include more than just the failure messages generated. The framework will not dictate in any way how the view will communicate validation failures to the user.

Any invalid values supplied by a user should be returned by the Business Object when requested. For example, if one has a Product Business Object with a Price property that can only accept numeric data, if a user provides the value "Bob" in the Price field of a form, when the Product object is returned to the view calling getPrice() will return the value "Bob".

Persistence Layer Agnostic

Although I am currently using Transfer, and have designed the framework to work with Transfer, that should be irrelevant to the framework. It should be possible to implement the framework, without making any modifications, into a model that uses any ORM or no ORM at all. The only requirement is that the model makes use of Business Objects.

MVC Framework Agnostic

It should be possible to implement the framework in any application using any MVC framework. That would include Fusebox, Model-Glue, Mach-II, ColdBox, and any homegrown MVC framework. This kind of goes without saying, as the framework is specific to the model layer of your application.

I believe that I have come up with an initial version of the framework that meets all of the above goals. I plan to write a number of posts explaining how I have designed this framework, why I made the choices I have made, and reviewing all of the code. Hopefully I will receive some feedback that will allow me to improve the framework, after which I hope to release it on RIAForge.