Getting Started with VT - Adding Validation Behaviour to Transfer Objects

Posted At : March 4, 2009 4:20 PM | Posted By : Bob Silverberg
Related Categories: ColdFusion, ValidateThis and Transfer, Transfer, ValidateThis

I have been working with my own validation framework, which I've named ValidateThis! (VT for short), for the past several months, and thought that it was time to talk about it some more, and provide some guidance to anyone who is interested in giving it a try. To that end I'm going to start a series about getting started with ValidateThis, specifically discussing using it to add validation behaviour to Transfer objects. The framework has been developed to allow for integration into any model that uses business objects, so Transfer is not a requirement to make use of VT, but as an example of integration with Transfer has already been done it seems like a sensible place to start.

As I haven't written about VT for quite some time I'll start with a brief introduction to VT and then move on to a practical guide to using VT with Transfer.

What is it?

VT is a set of ColdFusion components which enable a developer to add validation "smarts" into their Business Objects. The developer defines all of the validation business rules for an object in an xml file, and VT then automatically translates those into both client-side and server-side validation code. There is a lot more to it than that, but that is the foundation of what it does. If you want to see a demo of it in action, check it out at **www.validatethis.org**.

Why did I build it?

I built it because I wasn't happy with any of the approaches that I had previously used for implementing validation logic in my domain model. I decided to start by outlining the goals that I felt a validation framework should strive to fulfill.

What are those goals?

Here are the goals that I came up with:

- Flexibility
  - 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 an unlimited number of client-side validation implementations (e.g., jQuery, qForms, Prototype, homegrown, etc.).
- Extensibility
  - It should be possible to add new validation types without having to touch any of the existing framework code.
  - It should be possible to add new client-side validation implementations without having to touch any of the existing framework code.
- Code Generation
  - One should be able to define the validation rules as simple business rules, and the framework will generate all server-side and client-side validation code automatically.
  - The framework should be able to generate generic, but specific, validation failure messages, any of which can be overridden by an application developer.
- Flexible Feedback
  - The framework should return useful metadata back to the calling application, not just stock error messages. This will allow an application developer to choose how to implement that failure metadata in their view code.
  - Any invalid values supplied by a user should be returned by the Business Object when requested. For example, if one has a Product 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".
- Framework Agnostic
  - The framework should be persistence layer agnostic. 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 framework should be MVC framework agnostic. It should be possible to implement the framework in any application using any MVC framework.

I have written about these goals in the past. If you're interested more details can be found **here**.

Does it meet those goals?

I believe so. Obviously the more of a workout it gets the more I'll be able to evaluate that, and perhaps new goals will emerge.

What next?

Now that I've given an idea of what VT is and what it strives to be, I'll walk through the process of integrating it into an existing application that uses Transfer. Uncharacteristically I'm going to try to keep these posts short, just covering a single aspect of the integration in each article. I'm not sure that I'll be able to do that, but it's an admirable goal, don't you think?