

Contents

- [Overview](#)
- [Goals of the Framework](#)
 - [Flexible Validations](#)
 - [Code Generation](#)
 - [Flexible Feedback](#)
 - [Persistence Layer Agnostic](#)
 - [MVC Framework Agnostic](#)
- [Installation](#)
- [Requirements](#)

Overview

ValidateThis is a validation framework for ColdFusion objects and structures. It is designed to allow a developer to define the validation rules for their objects in a succinct fashion, and it will then automatically perform server-side validations and generate client-side validations.

Goals of the Framework

The ValidateThis framework is designed with the following goals in mind.

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 that currently ship with the framework are:

- Required - which includes three variations:
 - Simple - this property is always required
 - Dependent - this property is required if property x is populated
 - Dependent with Value - this property is required if property x has a value of y
 - Conditional - this property is required if condition x is true
- EqualTo - the value of this property must be equal to the value of property x
- Min - the value of this property must be at least x
- MinLength - the length of this property must be at least x
- Range - the value of this property must be between x and y
- RangeLength - the length of this property must be between x and y
- Numeric - the value of this property must be numeric
- Date - the value of this property must be a date
- Integer - the value of this property must be an integer
- Email - the value of this property must be an Email Address

- Regex - the value of this property must match the specified regex
- 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. The framework currently ships with a single JavaScript implementation:

- jQuery Validation Plugin

Code Generation

A developer should be able to define the validation rules in a simple manner in a single location, 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 end 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 property of a form, when the Product object is returned to the view, calling getPrice() will return the value "Bob". **Note:** A technique for achieving this is demonstrated in a sample application that ships with the framework, but the framework itself is not involved in "saving" this invalid data.

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 only requirement is that the model makes use of Business Objects. Note: The framework has been successfully implemented into applications using with Transfer, Reactor and ColdFusion 9's ORM features.

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, FW/1 and any homegrown MVC framework. This kind of goes without saying, as the framework is specific to the model layer of your application. Note: A Coldbox plugin has been developed which makes integration of the framework into Coldbox even simpler.

Installation

1. Download the source code from validatethis.riaforge.org.
2. Unzip the file into your webroot, or elsewhere and create a mapping called *ValidateThis* that points to it.

Note that the download comes with a number of sample applications, demonstrating ValidateThis being integrated into applications in various ways. None of those files and folders are required to make use of the framework - they are just examples.

Requirements

ValidateThis can be run on ColdFusion 8.0.1+, and on Railo 3.0.2+. It *may* run on OpenBD, but hasn't been tested.