ValidateThis! Setup

The zip file contains a demo application and the framework files.
Note that help is just a post away at http://groups.google.ca/group/validatethis or via email at bob.silverberg@gmail.com.

The Demo

To run the demo application you need:

- Transfer ORM (http://www.transfer-orm.com) v1.0 or higher
- Coldspring (http://www.coldspringframework.org) v.1.2
- A database. Scripts to create the required database objects for both SQL Server and MySQL can be found in the VTDemo folder, entitled mssql-setup.sql and mysql-setup.sql, respectively. Although the scripts only exist for SQL server and MySQL, they only create two tables, so it should be simple for anyone to create the required database and tables in any dbms that is supported by Transfer.
- Note that you'll need either a mapping for Transfer and Coldspring, or have them as sub-folders of the demo application.

To setup the demo:

1. Make sure you have met the requirements listed above.
2. Unzip all of the files from the download to the webroot from which you wish to run the demo.
3. Create a DSN in ColdFusion called VTDemo that points to the database that you created (see requirements above). You can change the DSN that the demo points to by editing /VTDemo/model/config/datasource.xml.cfm
4. Browse to the webroot in which you installed the demo and you should be able to see/run the demo.

Note that the commonassets and tags folders in the VTDemo directory hold the cfUniForm custom tag library developed by Matt Quackenbush which is **not** part of the framework. More info on that library can be found here:
http://cfuniform.riaforge.org/

The Framework Files

The framework files are in a sub-folder of the demo called ValidateThis. This makes it easier to just unpack and run the demo. If you want to integrate the framework with an existing model, just copy the contents of the /ValidateThis folder somewhere else on your machine and create a CF mapping to it, or copy it underneath the webroot of your application.

Integrating ValidateThis! With Your Model

Don't Panic:

If you don't feel like reading the following pages, which contain a pretty detailed description of what's required to get the framework working with your model, just take a look at the demo app. It's pretty straightforward and should give you all of the information that you need.

If you'd rather have a more detailed walkthrough of what's required, feel free to continue in with The Rest of the Story.

Also, in at attempt to limit the amount that you *must* read in order to get this working, each section below is broken into two parts. *Quick Steps* give you just the information you need, while *The Details* gives you a bit more information. *Quick Steps* assume you are using Transfer and Coldspring.

The Rest of the Story:

Note that the framework can be used without Transfer or Coldspring, but, as that's what I use, I only have a demo configured to work with those frameworks. For that reason I am going to explain how to get it working with that setup. You should be able to look at the demo app and figure out how to integrate it without Transfer quite easily. Getting it to work without Coldspring, on the other hand, will be a bit more work and will require a good understanding of Dependency Injection (DI). It is certainly possible, and not really that difficult, but if you do not have a compelling reason to do so I wouldn't venture down that path. If you do try it, however, I'd be happy to provide any assistance I can to get it working.

Getting Access to the Beans

Quick Steps

Add this line:

```
<import resource="/ValidateThis/config/Coldspring.xml.cfm" />
```

To the Coldspring config file for your application.

The Details

I am assuming that you are already using Coldspring as the DI engine for your app, and therefore that you already have your own Coldspring config set up for your model. That being the case, to add the framework to your bean factory, you can simply add an <import> statement to your own Coldspring config, like this:

```
<import resource="/ValidateThis/config/Coldspring.xml.cfm" />
```

This will allow all of the beans defined in the framework's CS config file to be available to your bean factory.

Of course, you can just copy the contents of VTDemo/config/Coldspring.xml.cfm into your own CS config file, if you prefer to have all of your beans in one place.

Defining the ValidateThisConfig Bean

Quick Steps

Add this bean definition to the Coldspring config file for your application:

```
<bean id="ValidateThisConfig"
class="coldspring.beans.factory.config.MapFactoryBean">
      <property name="sourceMap">
            <map>
                  <entry key="BOValidatorPath">
                        <value>BOValidator</value>
                  </entry>
                  <entry key="DefaultJSLib">
                        <value>jQuery</value>
                  </entry>
                  <entry key="JSRoot">
                        <value>js/</value>
                  </entry>
                  <entry key="defaultFormName">
                        <value>frmMain</value>
                  </entry>
            </map>
      </property>
</bean>
```

The Details

You can control some of the framework's behaviour and set default values using the ValidateThisConfig bean. More information about these settings can be found at http://www.silverwareconsulting.com/index.cfm/2009/3/8/ValidateThis-06--A-Whole-Bunch-of-New-Stuff.

For the most part, you can just leave the first two entries alone. For the JSRoot entry you should provide the path to your JavaScript files, and for the defaultFormName entry you should provide the name/id that you commonly use for your forms.

Creating the Validation Rules

Quick Steps

Create an xml file for each of your domain objects that require validations. Name the xml file the same as your Transfer class name (e.g., user.user.xml). Follow the schema found in /ValidateThis/validateThis.xsd. A sample file is available in /VTDemo/model/user.user.xml.

The Details

The preferred option for defining validation rules is to record them in an xml file. The framework comes with an XML Schema Definition file (/ValidateThis/validateThis.xsd) which describes the format of the XML file, and can be used in conjunction with your favourite XML editor for validation and code hinting. For a detailed explanation of the schema, refer to this blog posting: http://www.silverwareconsulting.com/index.cfm/2008/10/17/ValidateThis--Lets-Talk-Metadata

Create one xml file per domain object for which you wish to define validation rules. In my setup I use the transfer class name of my domain object as the name of the xml file, but you can theoretically call it anything you like.

You can find a sample xml file in the VTDemo directory in /model/user.user.xml

Note that a change is being considered which would move from one xml file per domain object to allowing a developer to define multiple domain objects in a single xml file (like transfer.xml).

Allowing the Framework to Find the XML Files

Quick Steps

The behaviour of the framework has been changed to that it will always look for your xml files in the same directory as your domain objects (i.e., Transfer decorators).

The Details

See above ;-)

Integrating the Framework with Your Domain objects

Quick Steps

Make your Transfer decorators resemble /VTDemo/model/AbstractTransferDecorator.cfc. Make your Coldspring beans for Transfer resemble the ones in /VTDemo/config/Coldspring.xml.cfm. You may need to look at *The Details* for this one.

The Details

Again, assuming a similar setup to the demo, you are using Transfer decorators as your domain objects. In that case integrating the framework is as easy as adding some methods to your abstract decorator (or all of your decorators if you're not using an abstract decorator – but why wouldn't you be?).

Open the file /model/AbstractTransferDecorator.cfc in the demo app and copy the following methods into your decorator:

- setup – note that you only need the setValidator() method – the setTheGateway() method has nothing to do with the framework
- onMissingMethod – note that if you already have an onMissingMethod in your decorator you'll have to integrate the code from this one into yours
- testCondition
- get/setValidationFactory
- get/setValidator
- get/setOnMMHelper
- wrapMe – not officially required, but required to take advantage of the ability to record invlaid values in the domain object
- get/setWrapper – not officially required, but required to take advantage of the ability to record invlaid values in the domain object

Some of these methods rely on Brian Kotek's excellent Bean Injector to inject singletons into the domain object (and to run setup()). That component is included in the download and is defined in the framework's Coldspring config file, so you will already have a copy of it.

You can find more into on Brian's component here:
http://coldspringutils.riaforge.org/

There are a number of ways of integrating the Bean Injector, but I find the easiest is to setup a certain configuration in Coldspring for Transfer and the Bean Injector. There is a sample Coldspring config for the demo app, which is located in VTDemo/config/Coldspring.xml.cfm. The following bean definitions are the ones that set up Transfer to use the Bean Injector (sorry for the line-wrap):

```
<bean id="transferFactory" class="transfer.TransferFactory">
      <constructor-arg
name="datasourcePath"><value>/model/config/datasource.xml.cfm</value></
constructor-arg>
      <constructor-arg
name="configPath"><value>/model/config/transfer.xml.cfm</value></constr
uctor-arg>
      <constructor-arg
name="definitionPath"><value>/TransferTemp</value></constructor-arg>
</bean>

<bean id="transfer" factory-bean="transferFactory" factory-
method="getTransfer" />

<bean id="TDOBeanInjectorObserver"
class="ValidateThis.util.TDOBeanInjectorObserver" lazy-init="false">
      <constructor-arg name="transfer"><ref bean="transfer"
/></constructor-arg>
      <constructor-arg
name="afterCreateMethod"><value>Setup</value></constructor-arg>
      <property name="beanInjector">
            <ref bean="beanInjector" />
      </property>
</bean>
```

So you'll want to mimic this setup in order to get ValidateThis integrated with your domain objects.

Optional Populate() Method

Quick Steps

This is optional. See *The Details* for more info.

The Details

There is one other method that I currently use in conjunction with my validation scheme and that's populate(). I do all of my datatype validations during populate, but that is not necessary for the framework to function. The only downside to not taking this approach is that you will lose access to the feature that "saves" values entered for properties that subsequently fail a datatype validation.

I understand that many developers already have some form of populate() in their arsenal, so the framework does not force you to change the way you're currently doing that.

If you do want to take advantage of the feature mentioned above, not only will you need to do something similar to what I'm doing in my populate method, you'll also need to "wrap" your domain object when it's first created. Take a look at the get() method in /model/server/AbstractService.cfc to see how it's done in the demo app.

Getting Client Side Validations to Work

Currently there is one implementation of client-side validations available, which uses the jQuery Validation Plugin.

In order to get these validations working in your app, you'll need to make sure that the following libraries are loaded:

jquery-1.3.2.min.js
jquery.field.min.js
jquery.validate.pack.js

Note that these files can be found in the /ValidateThis/client/jQuery/JS folder.

There is a method that will be available via your domain object called generateInitializationScript() which will generate all of the JavaScript statements required to load and configure those libraries. You can use that method, or, if you are already loading some of the libraries you can choose to do it manually. If you choose to do it manually it is suggested that you look at the source code for the demo to see what statements need to be there.

Boy, That Seems Like a Lot of Work

It's not really. I've done my best to document things well, which is why there is so much to read, but really, if you are trying to integrate this into a site that's already using Transfer and Coldspring, it shouldn't take that long. Defining your validation rules will take some time and effort, but the plumbing required to get the framework working with your model is actually quite minimal.

Need Help?

I am keen to get feedback on the framework, so I'm happy to help anyone attempting to use it. In the interest of keeping things DRY, I request that any questions/comments you have be directed to a Google group that I set up for the project. The group is located at http://groups.google.ca/group/validatethis