

## Contents

- [The ValidateThis Facade Object](#)
- [Creating the ValidateThis Facade Object](#)
  - [Creating the Object Manually](#)
  - [Creating the Object via Coldspring](#)
- [Using the ValidateThis Facade Object](#)
  - [The validate\(\) Method](#)
    - [Arguments](#)
    - [Return Value](#)
    - [Example](#)
  - [The getValidationScript\(\) Method](#)
    - [Arguments](#)
    - [Return Value](#)
    - [Example](#)
  - [The getInitializationScript\(\) Method](#)
    - [Arguments](#)
    - [Return Value](#)
    - [Examples](#)
  - [The getRequiredFields\(\) Method](#)
    - [Arguments](#)
    - [Return Value](#)
    - [Example](#)
  - [The getRequiredProperties\(\) Method](#)
  - [The getAllContexts\(\) Method](#)
    - [Arguments](#)
    - [Return Value](#)
    - [Example](#)
  - [The addRule\(\) Method](#)
  - [The newResult\(\) Method](#)
    - [Arguments](#)
    - [Return Value](#)
    - [Example](#)
  - [The getVersion\(\) Method](#)
    - [Arguments](#)
    - [Return Value](#)
    - [Example](#)

## The ValidateThis Facade Object

The ValidateThis Facade Object is an object that you create in your application code, and which you subsequently use to interact with the framework. It is a cfc named *ValidateThis.cfc*, which resides in the root of the *ValidateThis* folder.

## Creating the ValidateThis Facade Object

The ValidateThis Facade Object is designed to be created as a singleton, meaning there is one instance of it in your application and each request talks to the same instance. For that reason you can create an instance of the object manually, placing it into the application scope, or via an IOC container such as Coldspring.

### Creating the Object Manually

To create the object manually, just use `<cfoject>` or `createObject()` to create an instance. The most simple example would be:

```
<cfset application.ValidateThis = createObject("component", "ValidateThis.ValidateThis").init() />
```

When creating the object you can optionally pass in a [ValidateThisConfig Struct](#), which defines some of the behaviour of the framework. For example, if you wanted to specify the location of your JavaScript files and also the location of your [rule definition files](#), you could create a [ValidateThisConfig Struct](#) and pass it into the `init()` method of the facade object like so:

```
<cfset ValidateThisConfig = {JSRoot="/assets/js/", definitionPath="/myModel/"} />
<cfset application.ValidateThis = createObject("component", "ValidateThis.ValidateThis").init(ValidateThisConfig)/>
```

You would then use the facade object like so:

```
<cfset Result = application.ValidateThis.validate(myObject)/>
```

### Creating the Object via Coldspring

To create the object via Coldspring, you would define a bean in your `coldspring.xml` file for the facade like this:

```
<bean id="ValidateThis" class="ValidateThis.ValidateThis"/>
```

If you wish to pass some values to the framework via the [ValidateThisConfig Struct](#), you would define a bean for your `ValidateThisConfig` as a `coldspring.beans.factory.config.MapFactoryBean`, and then specify that bean in your `ValidateThis` bean, like so:

```
<bean id="ValidateThisConfig" class="coldspring.beans.factory.config.MapFactoryBean">
  <property name="sourceMap">
    <map>
      <entry key="definitionPath"><value>/myModel/</value></entry>
      <entry key="JSRoot"><value>/assets/js/</value></entry>
    </map>
  </property>
</bean>

<bean id="ValidateThis" class="ValidateThis.ValidateThis">
```

```
<constructor-arg name="ValidateThisConfig"><ref bean="ValidateThisConfig" /></constructor-arg>
</bean>
```

You would then use the facade object like so:

```
<cfset Result = application.beanFactory.getBean("ValidateThis").validate(myObject) />
```

## Using the ValidateThis Facade Object

Unless you are integrating the framework directly into your business objects, you use the framework by calling methods on the ValidateThis Facade Object. The following methods are available, listed in order of how often they might be used:

### The validate() Method

Use the `validate()` method to perform server-side validations on an object.

#### Arguments

It accepts the following arguments:

- *theObject* The actual object to validate. Note that this can also be a structure.
- *objectType (optional)* The name of the object type as defined to the framework. For more information on when this argument is optional, see the section on [Specifying the objectType on a Method Call](#).
- *context (optional)* The name of a context used in the object's rules definition file.
  - If passed, the object will be validated using the specified context. Only rules assigned to that context will be evaluated.
- *Result (optional)* A Result object that pre-exists.
  - If not passed in, a new Result object will be created and returned.
  - If an existing Result object is passed in, it will be appended to during this validation operation. This is useful when one needs to validate multiple objects during a given operation.
- *objectList (optional)* An array of objects that have already been validated. This is used internally by the `isValidObject` validator.
- *debuggingMode (optional)* Whether debugging should be turned on or not.
  - If debugging is on then debug information is gathered and returned via the Result object.
  - Valid values are *none* (the default), *strict* (which will throw an exception if an invalid xml file is encountered, and *on* which will log all debug information, including invalid xml files.
- *ignoreMissingProperties (optional)* Whether VT should ignore any properties for which rules have been defined but do not exist in the current object. The default is false, which means that missing properties will generate an exception.
- *locale (optional)* The locale to be used when generating default failure messages. Defaults to the `defaultLocale` value in the [ValidateThisConfig Struct](#).

#### Return Value

The `validate()` method returns a *Result* object, which can be interrogated and used to report validation failures. More details on the Result object can be found in the [Working with the Result Object](#) section.

#### Example

```
<cfset Result = application.ValidateThis.validate(myObject, 'User', 'Register') />
```

This would perform server-side validations on an object contained in the `myObject` variable, using a rules definition file called `User.xml`, and applying only those rules that belong to the context of `Register`. The results of the server-side validations would be returned in the *Result* object.

### The getValidationScript() Method

Use the `getValidationScript()` method to return JavaScript code for client-side validations.

#### Arguments

It accepts the following arguments:

- *theObject (optional)* The actual object for which to generate client-side validations.
- *objectType (optional)* The name of the object type as defined to the framework.
  - One of either *theObject* or *objectType* must be specified, so the framework can identify for which object the script should be returned.
  - For more information on why both are optional but at least one is required, see the section on [Specifying the objectType on a Method Call](#).
- *context (optional)* The name of a context used in the object's rules definition file.
  - If passed, the object will be validated using the specified context. Only rules assigned to that context will be evaluated.
- *formName (optional)* The name of the form in the html document to which validations are to be attached.
  - If not passed in the framework will use one of two values:
    - If a `formName` is associated with the given `context` in your [Rules Definition File](#) via the `<contexts>` element, that `formName` will be used.
    - Otherwise the `formName` that is specified in the `defaultFormName` key of the [ValidateThisConfig Struct](#) will be used.
- *JSLib (optional)* The name of the JavaScript implementation to be used to generate the script.
  - If not passed in the JSLib that is specified in the `DefaultJSLib` key of the [ValidateThisConfig Struct](#) will be used.
  - Note that as there is currently only one JS implementation available, it is currently unnecessary to pass this argument into this method.
- *locale (optional)* The locale to be used for translating validation failure messages.
  - Unless you are supporting multiple languages this argument can be ignored.

#### Return Value

The `getValidationScript()` method returns a string that contains an entire block of JavaScript which can then be inserted into your page. This includes the opening and closing `<script>` tags.

#### Example

```
<cfset theScript = application.ValidateThis.getValidationScript(objectType='User', context='Register') />
<cfhtmlhead text="#theScript#" />
```

This would ask the framework to generate the JavaScript required to perform the client-side validations defined in the rules definition file called `User.xml` for the context of `Register`. That script is then being loaded into the document via the `<cfhtmlhead>` tag.

### The getInitializationScript() Method

Use the `getInitializationScript()` method to return JavaScript code to set up client-side validations. This method is an optional, *helper* method. You can easily do the setup yourself, and there

are times when one might prefer to do some of the setup manually.

### Arguments

It accepts the following arguments:

- *JSLib* (optional) The name of the JavaScript implementation to be used to generate the script.
  - If not passed in the JSLib that is specified in the *DefaultJSLib* key of the [ValidateThisConfig Struct](#) will be used.
  - Note that as there is currently only one JS implementation available, it is currently unnecessary to pass this argument into this method.
- *JSLIncludes* (optional) A boolean indicating whether to return the JS statements that include the libraries required by the framework.
  - This exists to allow a developer to generate the setup script, but still manually add their own `<script>` tags for the libraries. This can be helpful when one is already including some of the libraries (e.g., jQuery) in their document.
- *locale* (optional) The locale to be used for translating validation failure messages.
  - Unless you are supporting multiple languages this argument can be ignored.

### Return Value

The `getInitializationScript()` method returns a string that contains an entire block of JavaScript which can then be inserted into your page. This includes the opening and closing `<script>` tags.

### Examples

```
<cfset theScript = application.ValidateThis.getInitializationScript()>
<cfhtmlhead text="#theScript#" />
```

This would ask the framework to generate the JavaScript required to setup the client-side validations. That script is then being loaded into the document via the `<cfhtmlhead>` tag.

The above example would generate the following JS:

```
<script src="/js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="/js/jquery.field.min.js" type="text/javascript"></script>
<script src="/js/jquery.validate.pack.js" type="text/javascript"></script>

<script type="text/javascript">
$(document).ready(function() {
  jQuery.validator.addMethod("regex", function(value, element, param) {
    var re = param;
    return this.optional(element) || re.test(value);
  }, jQuery.format("The value entered does not match the specified pattern ({0})");
});
</script>
```

If one does not want the framework to include the `<script>` tags, one would call it like this:

```
<cfset theScript = application.ValidateThis.getInitializationScript(JSIncludes=false) />
```

That would generate the following JS:

```
<script type="text/javascript">
$(document).ready(function() {
  jQuery.validator.addMethod("regex", function(value, element, param) {
    var re = param;
    return this.optional(element) || re.test(value);
  }, jQuery.format("The value entered does not match the specified pattern ({0})");
});
</script>
```

## The getRequiredFields() Method

Use the `getRequiredFields()` to return a struct that contains keys for each of the form fields that are required for a given object. This is useful if one wants to dynamically indicate those fields on a form, for example by placing an asterisk or an icon in front of each field.

### Arguments

It accepts the following arguments:

- *theObject* (optional) The actual object for which to generate client-side validations.
- *objectType* (optional) The name of the object type as defined to the framework.
  - One of either *theObject* or *objectType* must be specified, so the framework can identify for which object the fields should be returned.
  - For more information on why both are optional but at least one is required, see the section on [Specifying the objectType on a Method Call](#).
- *context* (optional) The name of a context used in the object's rules definition file.
  - If passed, the object will be validated using the specified context. Only rules assigned to that context will be evaluated.

### Return Value

The `getRequiredFields()` method returns a struct that contains one key for each required field.

### Example

```
<cfset theFields = application.ValidateThis.getRequiredFields(objectType='User', context='Register') />
```

This would ask the framework to return a struct of fields that are defined as required in the rules definition file called `User.xml` for the context of `Register`. That struct, called `theFields` in the above example, could then be interrogated when generating a form and would allow a developer to change the appearance of any required fields on the form.

## The getRequiredProperties() Method

This method is identical to the `getRequiredFields()` method, except it returns a struct of property names, rather than a struct of form field names. It is used internally by the framework and does not necessarily have a use case in your application code, but it is available if a use is discovered for it.

## The getAllContexts() Method

This method can be used to return all of the metadata about validations for a given object. It does not have a use case for application development, but can be very useful in debugging issues

with your validation rules.

### Arguments

It accepts the following arguments:

- *theObject* (optional) The actual object for which to generate client-side validations.
- *objectType* (optional) The name of the object type as defined to the framework.
  - One of either *theObject* or *objectType* must be specified, so the framework can identify for which object the fields should be returned.
  - For more information on why both are optional but at least one is required, see the section on [Specifying the objectType on a Method Call](#).

### Return Value

The *getAllContexts()* method returns a struct with one key per context, each of which contains the metadata for all rules defined for the object.

### Example

```
<cfset theFields = application.ValidateThis.getAllContexts(objectType='user') />
```

This would ask the framework to return a struct of metadata defined in the rules definition file called *User.xml*.

### The addRule() Method

The *addRule()* method can be used to dynamically add validation rules for an object. For more information on why you might choose to do that, and how to call the method, see the section on [Defining Validation Rules](#).

### The newResult() Method

The *newResult()* method can be used to return a new, empty *Result* object from the framework. This is generally unnecessary as the framework returns a *Result* object from all calls to *validate()*, but if a use case arises, this method is available.

### Arguments

This method accepts no arguments.

### Return Value

The *newResult()* method returns a *Result* object. More details on the *Result* object can be found in the [Working with the Result Object](#) section.

### Example

```
<cfset Result = application.ValidateThis.newResult (/>
```

### The getVersion() Method

The *getVersion()* method will return the version number of the framework. This can be useful when reporting problems with the framework.

### Arguments

This method accepts no arguments.

### Return Value

The *getVersion()* method returns a string giving you the number of the installed version.

### Example

```
<cfdump var="#application.ValidateThis.getVersion() #"/>
```