

## Contents

- [Internationalization with ValidateThis](#)
  - [ValidateThisConfig Keys for Translations](#)
    - [translatorPath](#)
    - [localeLoaderPath](#)
    - [localeMap](#)
    - [defaultLocale](#)
    - [Example](#)
  - [Performing Server-Side Validations](#)
  - [Generating Client-Side Validations](#)
  - [Creating Resource Bundles](#)
  - [Creating Custom Translators](#)

## Internationalization with ValidateThis

ValidateThis has been designed to be able to translate failure messages into multiple languages. By default this translation facility is turned off, but it can be enabled by specifying some additional keys in the [ValidateThisConfig Struct](#) when creating the ValidateThis Facade Object.

### ValidateThisConfig Keys for Translations

To enable automatic translations using resource bundles you use the following keys in your [ValidateThisConfig Struct](#) when you create the ValidateThis Facade Object:

#### translatorPath

This points to the component that will act as the translator for failure messages. Specify a value of `ValidateThis.core.RBTranslator` to enable the translator that makes use of resource bundles. If you'd prefer to use a different scheme (e.g., database-driven translations) then you can create your own translator object and provide the path to it.

#### localeLoaderPath

This points to the component that will load all of the resource bundles into the framework cache when it is instantiated. Specify a value of `ValidateThis.core.RBLocaleLoader` to enable the translator that makes use of resource bundles. As above, this can be overridden if you have your own way of loading resource bundles.

#### localeMap

This is a structure that contains information about which locales the application is to support, and their corresponding resource bundle property files. See below for an example.

#### defaultLocale

This optional key specifies the default locale that will be used by the framework if none is passed to method calls. By default it has the value of `en_US`, which is English.

#### Example

```
<cfset localeMap = {en_US="/rbs/en_US.properties",fr_FR="/rbs/fr_FR.properties"} />
<cfset validateThisConfig = {definitionPath="/demo/model/",jsRoot="/js/"} />
<cfset validateThisConfig.translatorPath = "ValidateThis.core.RBTranslator"/>
<cfset validateThisConfig.localeLoaderPath = "ValidateThis.core.RBLocaleLoader"/>
<cfset validateThisConfig.localeMap = localeMap/ >
<cfset application.ValidateThis = createObject("component", "ValidateThis.ValidateThis").init(validateThisConfig)/>
```

This would create an instance of the ValidateThis Facade Object in the application scope with support for English (`en_US`) and French (`fr_FR`).

### Performing Server-Side Validations

In order to generate default failure messages in the appropriate language, you'll need to pass the current locale into the call to `validate()`. Note that if the current request is using the default locale you do not have to pass that in, as the default value of the `locale` argument is the `defaultLocale` as defined in the [ValidateThisConfig Struct](#). After calling `validate()` and getting a Result object returned, when you call methods on the Result object you need to pass in the `locale` argument to the method in order to get failure messages back in the locale of your choosing. For example:

```
<cfset result = application.ValidateThis.validate(theObject=myObject, locale="fr_FR") />
<cfset failures = result.getFailures(locale="fr_FR") />
```

This would return an array of all validation failures in which each failure's message was translated into French.

### Generating Client-Side Validations

The only difference when generating client-side validations for a specific language is that you pass in a `locale` argument to the methods. For example:

```
<cfset initScript = application.ValidateThis.getInitializationScript(locale="fr_FR") />
<cfset theScript = application.ValidateThis.getValidationScript(theObject=myObject, locale="fr_FR") />
<cfhtmlhead text="#initScript#" />
<cfhtmlhead text="#theScript#" />
```

This would generate (and output via the calls to `cfhtmlhead`) the Javascript necessary for client-side validations, with failure messages in French.

### Creating Resource Bundles

When you create resource bundles for all of your supported languages, you should include keys for all of the default generated failure messages. These are included in the file `/ValidateThis/locales/en_US.properties`. If you do not do this then the framework will be unable to generate default failure messages. If you do not want to use any of the default messages (i.e., you specify a `failureMessage` attribute for each and every rule) then you do not need to worry about this.

## Creating Custom Translators

As mentioned above, the framework has been designed to allow you to override the Translator that is used to translate failure messages. The only Translator that is currently included in the framework is one that uses resource bundles. If you would like to use a different translation scheme (e.g., store translations in a database, use a webservice, etc.), you can create your own Translator.

Take a look at the BaseTranslator, RBTranslator, BaseLocaleLoader and RBLocaleLoader for guidance on producing your own. If you require any assistance please feel free to [contact us](#) for support.