Contents

ValidateThisConfig Struct

The framework is configured via a structure referred to as the ValidateThisConfig Struct. Each key has a default value in the framework, so if the default values match with your requirements you need not specify any keys. In that case you don't even have to define a ValidateThisConfig struct, as you can create the ValidateThis Facade Object without passing in a ValidateThisConfig struct.

The struct is composed of the following keys:

Core Settings

definitionPath

This key specifies the location of your rules definition files, one for each Business Object for which you want the framework to generate validations. By default it points to the */model/* folder.

- The value of *definitionPath* can be either a physical path to the folder (e.g., *C:\wwwroot\myApp\myModel*) or a relative path from either the web root or a CF mapping (e.g., */myModel/*).
- You can specify more than one folder to search by providing them as a comma delimited list.

For more information on how ValidateThis uses the *definitionPath* key, see the section [How ValidateThis Finds Your Rules Definition Files](#).

abstractGetterMethod

This key specifies the name of an abstract getter method if your objects use one. For example, some people design objects so that they have one method, e.g., *getValue*, which then accepts the name of a property as an argument. If your objects use an abstract getter you can specify the name of the getter via this key. The default value is *getValue*.

injectResultIntoBO

After server-side validations have been completed, VT can inject the Result object into your Business Object (BO), so that you can call *getVTResult* on your object to obtain the result. By default VT **will not** do this, but you can tell it to do so by setting this key to *true*.

defaultFailureMessagePrefix

When validations fail, both on the client-side and on the server-side, VT has a standard set of default messages that are generated. By default all of these messages are prepended with "The ", so a message might look like *The Email Address is required*. If you which to change this behaviour you can do so by providing a prefix via this key.

For example, if you set defaultFailureMessagePrefix="", the the message above would be generated as *Email Address is required*.

debuggingMode

There are three possible modes for debugging which is used in conjunction with the Result.getDebugging() method.

- none = debugging is disabled. This is the default
- info = evaluated rules / conditions and the outcome and available via the getDebugging method of the result
- strict = Same as info, with the addition of XML rules being validated for compliance against the validatethis.xsd

vtFolder

This key specifies the name of the physical folder in which ValidateThis is installed. By default it programmatically determines the name, so you generally do not have to set this, even if you store the framework in a folder other than *ValidateThis*. It may be necessary to change this value if you are on a case-sensitive system and have installed the framework in an alternate location.

Client-Side Validation Settings

jsRoot

The framework can be asked to generate all of the JavaScript statements necessary to load and configure all of the JS libraries required for client-side validations. This key specifies where the JavaScript files can be found on your web server. By default the location is *js/*.

Note that the bundled jQuery client-side validation scheme does not use this setting as the libraries required are pulled from CDNs. This key has been kept in case it is needed to support alternate Javascript implementations.

defaultFormName

This key specifies the default form name/id to be used when generating client-side validations. By default the form name is *frmMain*. This is provided for convenience as it is easily overridable on a case by case basis. More information on specifying the form name for your client-side validations is available in the [Generating Client-Side Validations](#) section.

jsIncludes

When calling the *getInitializationScript()* method to generate the JavaScript required to support client-side validations, you can pass a *JSIncludes* argument which tells VT whether to include statements loading the required JavaScript libraries. By default the value of JSIncludes is *true*, but if you would prefer to override that to *false* for your application you may do so using this key.

defaultJSLib

This key specifies which JavaScript implementation should be used for generating client-side validations. By default it specifies the *jQuery Validation Plugin,* which is the only JavaScript implementation that currently ships with the framework. Because ValidateThis can be configured with support for multiple JavaScript implementations, this key allows a developer to switch implementations globally. Because there is currently only one JS implementation, there should currently be no need to specify this key.

Note that this is referred to as a *Default* JSLib because the framework allows you to override this value on a case-by-case basis, effectively allowing you to use multiple JS implementations in a single application.

Component Paths

ValidateThis is constructed of objects, each of which serves a particular purpose. You can change and/or extend the behaviour of the framework by providing your own objects to VT, which would normally extend VT's own built-in objects. The following keys allow you to specify component paths which point to your own versions of certain objects.

resultPath

This key specifies the path to the Result component, which is what is returned to you after calling validate(). By default it points to *ValidateThis.util.Result*. If you want to provide your own custom Result object, then specify the component path to that object in this key.

boValidatorPath

This key specifies the path to the BOValidator component, which acts as a facade between your business object and the framework. By default it points to *ValidateThis.core.BOValidator*. It will generally by unnecessary to override this setting.

boComponentPaths

This key specifies a list of component paths in which your Business Objects (BOs) reside. This is used in conjunction with annotation support if you wish VT to pre-load a BOValidator for each of your objects.

extraRuleValidatorComponentPaths

This key specifies a list of component paths in which external ServerRuleValidators (SRVs) can be found. If you want to extend, override or add to the internal SRV components with your own application specific versions, use this key to provide the path(s).

extraClientScriptWriterComponentPaths

This key specifies a list of component paths in which external ClientScriptWriters (CSWs) can be found. If you want to extend, override or add to the internal client-side validation components, or even create a new JavaScript implementation, this is the key to use.

extraAnnotationTypeReaderComponentPaths

This key specifies a list of component paths in which external AnnotationTypeReaders (ATRs) can be found. This will be used if you want to add support for a new annotation type. Currently JSON, XML and VTML (coming soon) are supported.

extraFileReaderComponentPaths

This key specifies a list of component paths in which external FileReaders (FRs) can be found. This will be used if you want to add support for a new external file type. Currently XML and JSON are supported. Note that this needs to be used in conjunction with the *ExternalFileTypes* key, discussed below.

externalFileTypes

This key specifies a list of external file types supported by the framework. The default value is *xml,json*. If you are adding support for a new file type via the *ExtraFileReaderComponentPaths*, you'll need to add that new type to this list as well.

Internationalization Keys

The following keys are used to configure the Internationalization (i18n) behaviour of the framework. If you are using the framework with a single language you can safely leave them out of your ValidateThisConfig struct. More information about i18n is available in the [Internationalization with ValidateThis](#) section.

translatorPath

This key specifies the path to the Translator component, which is used to translate validation failure messages. By default it points to the framework's BaseTranslator.cfc, which doesn't actually translate anything. Therefore, unless you need to support multiple languages this key does not need to be specified.

localeLoaderPath

This key specifies the path to the LocaleLoader component, which is used to load information about various language translations into the Translator. By default it points to the framework's BaseLocaleLoader.cfc, which doesn't load anything. Again, unless you need to support multiple languages this key does not need to be specified.

localeMap

This key provides a struct which contains information about which locales the application is to support, and their corresponding resource bundle property files. Prior to version 1.0 this was an empty struct, but now it must contain at least the path to the resource bundle that contains the default failure messages, so the current default value is *{en_US='/ValidateThis/locales/en_US.properties'}*. Unless you need to support multiple languages you can leave this key untouched.

defaultLocale

This 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". Unless you need to support multiple languages this key does not need to be specified.