Registration Validation 1.0 Component Specification

1. Design

This component provides an implementation of a validation plug-in for the Registration Services component. This plug-in performs all validation needed to make sure a registration will succeed if no runtime problems occur (such as a database crash).

Also it will be modeled in such a way that each piece of validation could be removed by configuration or new pieces added.

The DataValidationRegistrationValidator that implements the RegistrationValidator will delegate all validation to ConfigurableValidator, which is built upon the validators in Data Validation component. These are also in many cases extended to provide configuration and logging support for all methods.

Messages returned from the validation are customizable using the Document Generator component.

This design makes use of the Logging Wrapper to facilitate informational and debugging logging in the services implementation. Logging is not performed inside bean implementations as there is no need to know such granular and trivial information. Logging is optional.

Note: When viewing the Poseidon ZUML documentation, an "empty" String means a String that, after being trimmed, has a length of zero.

1.1 Design considerations

1.1.1 EJBs, ConfigManager, and Logging Wrapper

The EJB specification stipulates that File I/O is not allowed during the execution of the bean. At first glance this might mean that the use of the ConfigManager, and by extension Object Factory, could not be used because it performs property retrieval and storing using files. However, the restriction is only placed on the bean's lifetime, not the ConfigManager's. Therefore, as long as the ConfigManager does not perform I/O itself during the execution of the bean, or to be more accurately, that the thread performing a bean operation does not cause the ConfigManager to perform I/O, the use of ConfigManager to hold an in-memory library of properties is acceptable. Therefore, this design makes extensive use of ConfigManager and Object Factory, again, with the stipulation that the ConfigManager implementation does not perform I/O when this component is retrieving properties.

The issue of using Logging Wrapper is similar. The user of this component must ensure that logging is performed in a manner that does not violate EJB standards, such as not writing to a file or console.

1.2 Design Patterns

1.2.1 Strategy

The DataValidationRegistrationValidator uses ConfigurableValidator as a **Strategy**.

1.2.2 Composite

Various validators, such as AndValidator and OrValidator, are made up of other validators as per the **Composite** pattern. This component doesn't really introduce it as much as its being here is a consequence of having to provide extensions for the composite validators to support configuration with a RegistrationValidator, but it does take advantage of it for each composite validator to call its children's setRegistrationValidator method when its own is called, so a single call to the root validator will ensure that all validators are configured.

1.3 Industry Standards

- EJB (specifically, to ensure this component is compatible with EJB restrictions, as defined in http://java.sun.com/blueprints/qanda/ejb_tier/restrictions.html).
- JavaBeans

1.4 Required Algorithms

1.4.1 Logging

This section is the central place to describe how logging is performed in lieu of stating this in each method in the ZUML.

All methods in DataValidationRegistrationValidator and every validator have access to a Log and should log in the following manner:

- Method entrance and exit at INFO level.
- All exceptions and errors at ERROR level. This includes illegal arguments.
- All calls to external TopCoder classes (just to managers and services classes, not beans) at DEBUG level. This includes before the call and after the call.
- All return value of the getMessage(Object) at DEBUG level..

Note that if the DataValidationRegistrationValidator 's log is null, no logging is to be performed.

1.4.2 Template XML Data Generation

Most validations require the generation of a message if validation fails. Each message will be created/filled using the Document Generator component. Part of the task will be to create the required layered XML that will pass the data to the

messages. This section will define the structure that each validator that uses messages will adhere to.

Each such validator will perform the following steps to create the message:

```
1. Get instance of DocumentGenerator: DocumentGenerator.getInstance()
2. get Template with
   documentGenerator.getTemplate(this.getValidationMessage())
3. build XML template data (See CS 1.4.2.5)
4. Generate message text:  documentGenerator.applyTemplate(template,data)
```

As can be seen, the message in the validator is used as the template name in the DocumentGenerator.

The following sections will show the XML formats that the developer will assemble. Simple string concatenation will be sufficient. Note that all the information will be available.

At minimum, the following standard information is made available, corresponding to the ValidationInfo stuff. The following contains some sample data:

```
<DATA>
    <USER>
        <HANDLE>jjones</HANDLE>
        <FIRST_NAME>jim</FIRST_NAME>
        <LAST_NAME>Jones</LAST_NAME>
        <EMAIL>jim.jones@topcoder.com</EMAIL>
       <ALTERNATE_EMAIL>
            <EMAIL_VALUE>jim@tc.com</EMAIL_VALUE>
        </ALTERNATE_EMAIL>
       <ALTERNATE_EMAIL>
            <EMAIL_VALUE>jack@tc.com</EMAIL_VALUE>
        </ALTERNATE_EMAIL>
        <DESIGN_RATING>2000</ DESIGN_RATING>
        <DESIGN_RELIABILITY>80</DESIGN_RELIABILITY>
        <DESIGN_VOLATILITY>345</DESIGN_VOLATILITY>
        <DESIGN_NUM_RATINGS>1</ DESIGN_NUM_RATINGS>
        <DEV_RATING>1234</ DEV_RATING>
        <DEV_RELIABILITY>60</DEV_RELIABILITY>
        <DEV_VOLATILITY>123</DEV_VOLATILITY>
        <DEV_NUM_RATINGS>1</ DEV_NUM_RATINGS>
    </USER>
    <FULL_PROJECT_DATA>
        <RESOURCE>
            <ROLE>Team Captain</ROLE>
        </RESOURCE>
        <PROJECT_HEADER>
            <NAME>Team Management</NAME>
            <CATEGORY>Design</CATEGORY>
            <STATUS>Active</STATUS>
        </PROJECT_HEADER>
        <TEAM_HEADER>
            <NAME>Team TAG</NAME>
            <FINALIZED>false</FINALIZED>
            <CAPTAIN_RESOURCE_ID>1</CAPTAIN_RESOURCE_ID>
            <CAPTAIN_PAYMENT_PERCENTAGE>50</CAPTAIN_PAYMENT_PERCENTAGE>
            <DESCRIPTION>This is team TAG</DESCRIPTION>
        </TEAM_HEADER>
        <TECHNOLOGY>
            <NAME>Java</NAME>
        </TECHNOLOGY>
```

```
        <TECHNOLOGY>
            <NAME>SQL</NAME>
        </TECHNOLOGY>
    </FULL_PROJECT_DATA>
    :
    : // custom data
    :
<DATA>
```

1.4.2.1 MemberNotRegisteredWithRoleForProjectValidator

This validator will provide the following additional data.

```
<DATA>
    :
    : // standard data
    :
    <ROLE_ID>1</ROLE_ID>
<DATA>
```

1.4.2.2 MemberNotTeamCaptainWithMembersForProjectValidator

This validator will provide the following additional data.

```
<DATA>
    :
    : // standard data
    :
    <TEAM_CAPTAIN_ROLE>Team Captain</TEAM_CAPTAIN_ROLE>
<DATA>
```

1.4.2.3 MemberNotExceededMaxProjectRegistrationLimitValidator

This validator will provide the following additional data.

```
<DATA>
    :
    : // standard data
    :
    <MAX_REGISTRATION_COUNT>5</MAX_REGISTRATION_COUNT>
<DATA>
```

1.4.2.4 MemberNotTeamMemberForProjectValidator

This validator does not provide any additional custom data

1.4.2.5 MemberNotBarredValidator

This validator does not provide any additional custom data

1.4.2.6 ProjectOfTypeValidator

This validator will provide the following additional data.

```
<DATA>
    :
    : // standard data
    :
    <PROJECT_TYPE>Design</PROJECT_TYPE>
<DATA>
```

1.4.2.7 ProjectOfCategoryValidator

This validator will provide the following additional data.

```
<DATA>
    :
    : // standard data
    :
    <PROJECT_CATEGORY>Mathematical</PROJECT_CATEGORY>
<DATA>
```

1.4.2.8 ProjectInPhaseValidator

This validator will provide the following additional data.

```
<DATA>
    :
    : // standard data
    :
    <PHASE>1</PHASE>
<DATA>
```

1.4.2.9 MemberMinimumRatingForRatingTypeValidator

This validator will provide the following additional data.

```
<DATA>
    :
    : // standard data
    :
    <MINIMUM_RATING>2000</MINIMUM_RATING>
    <RATING_TYPE>Design</RATING_TYPE>
<DATA>
```

1.4.2.10 MemberMinimumReliabilityForRatingTypeValidator

This validator will provide the following additional data.

```
<DATA>
    :
    : // standard data
    :
    <MINIMUM_RELIABILITY>80.0</MINIMUM_RELIABILITY>
    <RATING_TYPE>Design</RATING_TYPE>
<DATA>
```

1.4.2.11 MemberMinimumVolatilityForRatingTypeValidator

This validator will provide the following additional data.

```
<DATA>
    :
    : // standard data
    :
    <MINIMUM_VOLATILITY>300</MINIMUM_VOLATILITY>
    <RATING_TYPE>Design</RATING_TYPE>
<DATA>
```

1.4.2.12 MemberMinimumNumberOfRatingsForRatingTypeValidator

This validator will provide the following additional data.

```
<DATA>
    :
    : // standard data
    :
    <MINIMUM_NUM_RATINGS>1</MINIMUM_NUM_RATINGS>
    <RATING_TYPE>Design</RATING_TYPE>
<DATA>
```

1.4.2.13 NotValidator

This validator does not provide any additional custom data

1.5 Component Class Overview

1.5.1 com.topcoder.registration.validation

This package holds the main interfaces and classes of this component.

DataValidationRegistrationValidator
Implementation of the RegistrationValidator interface. This implementation makes use of a large array of components to accomplish its task of validating data: The Team Manager to get team information, Project Services to get project information, and Ban Manager to get ban information

To provide a good view as the steps are progressing in each method, as well as full debug information for developers, the Logging Wrapper component is used in each method. To configure this component, the ConfigManager and ObjectFactory components are used.

Also provided are configuration-backed and programmatic constructors. This allows the user to either create all internal supporting objects from configuration, or to simply pass the instances directly.

Thread Safety: This class is immutable but operates on non thread safe objects, thus making it potentially non thread safe.

ConfigurableValidator
An extension of the Data Validation's Object Validator to add a setter for the DataValidationRegistrationValidator so each validator in this component has access to managers, services, and logger. This allows for one centralized are for such configuration. All implemented validators in this component implement this interface, and all new validators will have to as well.

Thread Safety: In general, implementations should be thread-safe, but there is no need to synchronize the id field.

AbstractConfigurableValidator
An adapter for all simple and conditional validators that implements most of the methods in the ConfigurableValidator. Most of the work is done by the extended

AbstractObjectValidator from Data Validation. This extension allows the simple and conditional validators to concentrate on the getMessage validation method to perform the specific validation. Future custom validators can take advantage of this adapter.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

ValidationInfo
This class is a DTO that wraps the passed registration data so it can be passed as one object to the validators. This class implements java.io.Serializable.

This entity follows java bean conventions for defining setters and getters for these properties, as well as the minimum empty constructor. A full constructor is also provided for convenient usage.

Thread Safety: This class is mutable and not thread safe.

OperationResultImpl
Simple implementation of the OperationResult interface from the Team Services component. Implements all methods in that interface, and adds the required setters. It provides the required default constructor if the user wants to set all values via the setters, and one full constructor to set these values in one go.

Thread Safety: This class is mutable and not thread safe.

1.5.2 com.topcoder.registration.validation.validators.util

This package holds extensions to some Data Validation validators so they can be used with ConfigManager

NullValidator
A wrapper around the Data Validation's NullValidator to provide logging in the methods.

This class implements ConfigurableValidator so it can be set with the DataValidationRegistrationValidator manager so it can obtain the Log to perform logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

NotValidator

A replacement of the Data Validation's NotValidator so the inner validator can be set from configuration. It behaves exactly the same way.

This class implements ConfigurableValidator so it can be set with the DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

AndValidator
A wrapper around the Data Validation's AndValidator to provide logging in the methods and allow the aggregated validators to be created from configuration. It also allows for configuration of a flag to have validation stop after the first error, or to provide non-short-circuited validation.

This class implements ConfigurableValidator so it can be set with the DataValidationRegistrationValidator manager so it can obtain the Log to perform logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

OrValidator
A wrapper around the Data Validation's OrValidator to provide logging in the methods and allow the aggregated validators to be created from configuration. It also allows for configuration of a flag to have validation stop after the first error, or to provide non-short-circuited validation.

This class implements ConfigurableValidator so it can be set with the DataValidationRegistrationValidator manager so it can obtain the Log to perform logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

1.5.3 com.topcoder.registration.validation.validators.simple

This package holds the implemented simple validators

MemberNotRegisteredWithRoleForProjectValidator
A simple validator that checks if a member is not registered with a given role for a project.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

MemberNotTeamCaptainWithMembersForProjectValidator
A simple validator that checks that a member is not a captain of a team that has other team members in positions. Team Manager is used here.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

MemberNotExceededMaxProjectRegistrationLimitValidator
A simple validator that checks if a member is not registered with a given role for a project. Project Services is used here.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

MemberNotTeamMemberForProjectValidator
A simple validator that checks that a member is not a member of any team, as a member or a captain.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method.

The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

MemberNotBarredValidator
A simple validator that checks that a member has not exceeded the maximum amount of projects one can register for. This is done by checking with the BanManager.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

ProjectOfTypeValidator
A simple validator that checks that the project is of a given type.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

ProjectOfCategoryValidator
A simple validator that checks that the project is of a given category.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

ProjectInPhaseValidator
A simple validator that checks that the project is in a certain phase.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

MemberMinimumRatingForRatingTypeValidator
A simple validator that checks if the user has a minimum required rating.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

MemberMinimumReliabilityForRatingTypeValidator
A simple validator that checks if the user has a minimum required reliability.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

MemberMinimumVolatilityForRatingTypeValidator
A simple validator that checks if the user has a minimum required volatility.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

MemberMinimumNumberOfRatingsForRatingTypeValidator
A simple validator that checks if the user has a minimum required number of ratings.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

1.5.4 com.topcoder.registration.validation.validators.conditional

This package holds the implemented conditional validators

ProjectTypeConditionalValidator
A conditional validator that checks the registration is for a project with a given type. The validation only proceeds to an inner validator if it is.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

ProjectCategoryConditionalValidator
A conditional validator that checks if the project is of a given category to see if validation should be performed. The validation only proceeds to an inner validator if it is.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

ProjectIdentifierConditionalValidator
A conditional validator that checks if the project is of a given ID to see if validation should be performed. The validation only proceeds to an inner validator if it is.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

RegisteringResourceRoleConditionalValidator
A conditional validator that checks if the registering role is of a given ID to see if validation should be performed. The validation only proceeds to an inner validator if it is.

This class extends AbstractConfigurableValidator that implements most of the validation methods, so this class only needs to implement the getMessage method. The DataValidationRegistrationValidator manager, available via the getRegistrationValidator method, provides all necessary managers and services for this validator. It also provides the Log for logging.

Thread Safety: This class is mutable and thus is not thread-safe, but the actual properties and managers used to validate are immutable, so this class will be effectively thread-safe in the context of its usage in this component.

1.6 Component Exception Definitions

This component defines three custom exceptions. Note that all are runtime exceptions.

ValidationProcessingException

Extends RegistrationValidationException. Called by all validation methods in case an unexpected system error occurs during validation.

RegistrationValidationConfigurationException
Extends RegistrationValidationException. Called by all namespace constructors in all validators if a configuration-related error occurs, such as a namespace not being recognized by ConfigManager, or missing required values, or errors while constructing the managers and services and validators.

1.7 Thread Safety

Thread safety is mentioned as a required part of this component. The component is virtually thread-safe, and under expected conditions, it is effectively thread-safe.

All object validator implementations are not thread-safe as they are based on the Data Validation classes that are mutable and not thread safe. If the validators had to be made thread-safe, it would either have to be done at the Data Validation level or the validators would have to be changed to not use the Data Validation.

In the context of the expected usage, the validators are thread-safe, as they will not be mutated by more than one registration validator instance, and it is also expected that none of the value objects passed for validation will be used or mutated by more than one thread at a time.

Since the registration validator and the inner validators does not mutate any external state such as a persistence store, transactional control is not a factor in this component.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.4
- Compile target: Java 1.4

2.2 TopCoder Software Components

- Configuration Manager 2.1.5
    - Used for configuration
- Object Factory 2.0.1
    - Used to obtain instances of required objects. Some of the required objects are instances of TeamManager, BanManager, etc…
- Base Exception 2.0
    - TopCoder standard for all custom exceptions.
- Logging Wrapper 2.0

    - Used for logging actions in all validators via the DataValidationRegistrationValidator. A Log instance is obtained from LogManager.
- Project Services 1.0
    - Provides the ProjectServices to obtain detailed project information
- Ban Management 1.0
    - Provides the BanManager to check if members are banned
- Registration Services 1.0
    - The origin of the RegistrationValidator and RegistrationInfo entities.
- User Project Data Store 2.0
    - Provides ExternalUser information.
- Data Validation 2.0
    - Used as the backbone for the validators. All extend ObjectValidator.
- Team Management 1.0
    - Provides team information via the TeamManager interface
- Document Generator 2.0
    - Provides the facility to generate messages.
- Team Services
    - Provides the OperationResult interface.
- Project Phases 2.0
    - Provides the phase model: Project and Phase.
- Resource Management 1.1
    - Provides the classes Resource, and ResourceRole.
- Phase Management 1.1
    - Provides the Project class

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

There are no third party components that need to be used directly by this component.

3. Installation and Configuration

3.1 Package Names

com.topcoder.registration.validation
com.topcoder.registration.validation.validators.simple
com.topcoder.registration.validation.validators.conditional
com.topcoder.registration.validation.validators.util

3.2 ConfigurationParameters

3.2.1 DataValidationRegistrationValidator

Parameter	Description	Details
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required.	Example: "com.topcoder.specify"
validatorKey	Key for the ConfigurableValidator to pass to ObjectFactory. Required	Valid key
teamManagerKey	Key for the TeamManager to pass to ObjectFactory. Required	Valid key
projectServicesKey	Key for the ProjectServices to pass to ObjectFactory. Required	Valid key
banManagerKey	Key for the BanManager to pass to ObjectFactory. Required	Valid key
loggerName	The name of the log to get from the LogManager. Optional	A valid name of the log. If not given, logging is not performed.

3.2.2 Implemented validators

All validators can be created from configuration. All simple validators will at least have the following properties. Other validators will not.

The bundle will hold the name of the template to use from DocumentGenerator.

Parameter	Description	Details
bundleName	The name of the bundle. Required if defaultMessage is null.	"myBundle"
bundleLocaleLanguage	The language portion of the Locale to use with bundle. Optional.	"en" Use default locale if not given
bundleLocaleCountry	The language portion of the Locale to use with bundle. Optional and only applicable if bundleLocaleLanguage is provided.	"ca"
bundleLocaleVariant	The variant portion of the Locale to use with bundle. Optional and only applicable if bundleLocaleCountry is provided.	"Traditional_WIN"
messageKey	The key to use to get message (a.k.a. the temp;late name) from resource bundle. Required if bundleName is used.	A non-null/empty key
defaultMessage	The default template name to use if there is no bundle or if the bundle does not have an entry.	"myTemplate"

	Required.	

3.2.2.1 MemberNotRegisteredWithRoleForProjectValidator

This is a simple validator, so it will have the properties in 3.2.2.
It has also the following property.

Parameter	Description	Details
roleId	The role ID that the member should not have. Required.	1

3.2.2.2 MemberNotTeamCaptainWithMembersForProjectValidator

This is a simple validator, so it will have the properties in 3.2.2.
It has also the following property.

Parameter	Description	Details
teamCaptainRoleId	The ID of the team captain role. Required.	1

3.2.2.3 MemberNotExceededMaxProjectRegistrationLimitValidator

This is a simple validator, so it will have the properties in 3.2.2.
It has also the following property.

Parameter	Description	Details
maxRegistrationCount	The maximum number of projects the member can be registered for. Required.	2

3.2.2.4 MemberNotTeamMemberForProjectValidator

This is a simple validator, so it will have the properties in 3.2.2.
It has no additional properties.

3.2.2.5 MemberNotBarredValidator

This is a simple validator, so it will have the properties in 3.2.2.
It has no additional properties.

3.2.2.6 ProjectOfTypeValidator

This is a simple validator, so it will have the properties in 3.2.2.
It has also the following property.

Parameter	Description	Details
projectTypeId	The type ID the project should have. Required.	1

3.2.2.7 ProjectOfCategoryValidator

This is a simple validator, so it will have the properties in 3.2.2.

It has also the following property.

Parameter	Description	Details
projectCategoryId	The project category ID the project should have. Required.	1

3.2.2.8 ProjectInPhaseValidator

This is a simple validator, so it will have the properties in 3.2.2.
It has also the following property.

Parameter	Description	Details
phaseId	The ID of the phase the project should be in. Required.	1

3.2.2.9 MemberMinimumRatingForRatingTypeValidator

This is a simple validator, so it will have the properties in 3.2.2.
It has also the following properties.

Parameter	Description	Details
minimumRating	The minimum rating the user must have for the given rating type. Required.	2000
ratingType	The rating type that will be validated. Required. This must be a valid RatingType.	“Design”

3.2.2.10 MemberMinimumReliabilityForRatingTypeValidator

This is a simple validator, so it will have the properties in 3.2.2.
It has also the following properties.

Parameter	Description	Details
minimumReliability	The minimum reliability the user must have for the given rating type. Required.	80.0
ratingType	The rating type that will be validated. Required. This must be a valid RatingType.	“Design”

3.2.2.11 MemberMinimumVolatilityForRatingTypeValidator

This is a simple validator, so it will have the properties in 3.2.2.
It has also the following properties.

Parameter	Description	Details
minimumVolatility	The minimum volatility the user must have for the given rating type. Required.	300
ratingType	The rating type that will be validated. Required. This must be a valid RatingType.	“Design”

3.2.2.12 MemberMinimumNumberOfRatingsForRatingTypeValidator

This is a simple validator, so it will have the properties in 3.2.2.

It has also the following properties.

Parameter	Description	Details
minimumNumRatings	The minimum number of ratings the user must have for the given rating type. Required.	1
ratingType	The rating type that will be validated. Required. This must be a valid RatingType.	“Design”

3.2.2.13 ProjectTypeConditionalValidator

This is a conditional validator. It has the following properties.

Parameter	Description	Details
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required.	Example: “com.topcoder.specify”
innerValidator	Key for the ConfigurableValidator to pass to ObjectFactory. Required	Valid key
projectTypeId	The type ID the project should have for validation to occur. Required.	1

3.2.2.14 ProjectCategoryConditionalValidator

This is a conditional validator. It has the following properties.

Parameter	Description	Details
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required.	Example: “com.topcoder.specify”
innerValidator	Key for the ConfigurableValidator to pass to ObjectFactory. Required	Valid key
projectCategoryId	The category ID the project should have for validation to occur. Required.	1

3.2.2.15 ProjectIdentifierConditionalValidator

This is a conditional validator. It has the following properties.

Parameter	Description	Details
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required.	Example: “com.topcoder.specify”
innerValidator	Key for the ConfigurableValidator to pass to ObjectFactory. Required	Valid key
projectId	The ID the project should have for validation to occur. Required.	1

3.2.2.16 RegisteringResourceRoleConditionalValidator

This is a conditional validator. It has the following properties.

Parameter	Description	Details
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required.	Example: “com.topcoder.specify”
innerValidator	Key for the ConfigurableValidator to pass to ObjectFactory. Required	Valid key

roleId	The ID the registering role should have for validation to occur. Required.	1

3.2.2.17 NotValidator

This is a utility validator, and it will have the properties in 3.2.2.
It has also the following properties.

Parameter	Description	Details
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required.	Example: "com.topcoder.specify"
innerValidator	Key for the ConfigurableValidator to pass to ObjectFactory. Required	Valid key

3.2.2.18 AndValidator and OrValidator

This is a utility validator. It has also the following properties.

Parameter	Description	Details
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required.	Example: "com.topcoder.specify"
innerValidator#	Key for the ConfigurableValidator to pass to ObjectFactory. Required There will be 1..n ordered keys, with a numbered, consecutive 1-based suffix for each key.	Valid key
shortCircuited	A flag as to whether validation should stop on first error, or continue accumulating messages until all validators are asked. Defaults to true.	"true"/"false"

3.3 Dependencies Configuration

3.3.1 TopCoder dependencies

The developer should refer to the component specification of the TopCoder components used in this component to configure them. Please see section 2.2 for a list of these components.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

None

4.3 Demo

4.3.1 Setup

Much of the setup for any demo involves properly configuring all the components. This demo will not go into that depth, but it will show a sample configuration XML file.

Here we define the configuration parameters in ConfigManager:

```
<Config name="TestConfig">
   <Property name="specNamespace">
         <Value>SpecificationNamespace</Value>
   </Property>
   <Property name="validatorKey">
         <Value>validatorKey</Value>
   </Property>
   <Property name=" projectServicesKey ">
         <Value> projectServicesKey </Value>
   </Property>
   <Property name=" banManagerKey ">
         <Value> banManagerKey </Value>
   </Property>
   <Property name="teamManagerKey">
         <Value>teamManagerKey</Value>
   </Property>
   <Property name="loggerName">
         <Value>defaultLogger</Value>
   </Property>
</Config>
```

The construction of the registration validator class would be done in the following manner.

```
// Create RegistrationValidator from configuration.
RegistrationValidator registrationValidator = new
DataValidationRegistrationValidator (“testDemoConfig”);
```

4.3.2 Usage

A typical usage scenario involves validation of user information: Suppose we have validation to check that the user is not on any team on the project, has a rating of 1300, and the project is still in registration:

Our configuration would then use the following validators:

- MemberNotTeamMemberForProjectValidator
- MemberMinimumRatingForRatingTypeValidator
- ProjectInPhaseValidator

We can then test the suitability of various users for a position on the project:

We use the service instance and scenario setup from 4.3.1.

```
// Create FullProjectData for test.
FullProjectData project = createFullProjectDataForTest();
```

```
// sets the user information and registration information for the user1.
// the user 1 is not registered, has rating 900, and the project is in
// registration
RegistrationInfo info1 = new RegistrationInfoImpl();
info1.setProjectId(2);
info1.setRoleId(3);
info1.setUserId(2);
RatingInfo ratingInfo1 = new RatingInfo(RatingType.DESIGN, 900, 5, 20);
ExternalUser user1 = new ExternalUserImpl(1, "handle1", "user1",
        "TCMember", "user1@topcoder.com");
((ExternalUserImpl) user1).addRatingInfo(ratingInfo1);

OperationResult result1 = registrationValidator.validate(info1, user1,
        project);
// The result will be that validation fails because the rating is too
// small

// sets the user information and registration information for the user2.
// the user 2 is already registered on this project, has rating 2000,
// and the project is in registration
RatingInfo ratingInfo2 = new RatingInfo(RatingType.DESIGN, 2000, 5, 20);
ExternalUser user2 = new ExternalUserImpl(2, "handle2", "user2",
        "TCMember", "user2@topcoder.com");
((ExternalUserImpl) user2).addRatingInfo(ratingInfo2);
RegistrationInfo info2 = new RegistrationInfoImpl();
info2.setProjectId(2);
info2.setRoleId(3);
info2.setUserId(1);
OperationResult result2 = registrationValidator.validate(info2, user2, project);
// The result will be that validation fails because the user is
// already registered on this project

// sets the user information and registration information for the user3.
// the user3 is not registered, has rating 1345, and the project is in
// registration
RegistrationInfo info3 = new RegistrationInfoImpl();
info3.setProjectId(2);
info3.setRoleId(3);
info3.setUserId(3);
RatingType ratingType3 = RatingType.DESIGN;
RatingInfo ratingInfo3 = new RatingInfo(RatingType.DESIGN, 1345,
        5, 20);
ExternalUser user3 = new ExternalUserImpl(3, "handle3", "user3",
        "TCMember", "user3@topcoder.com");
((ExternalUserImpl) user3).addRatingInfo(ratingInfo3);

OperationResult result3 = registrationValidator.validate(info3, user3,
        project);
// The result will be that validation succeeds

// This demo has provided a typical scenario for the usage of the
validator.
```

4.3.3 Validator assembly

The above demo shows how the registration validator validates using configured validators. However, how does one decide which validators are applicable? Since the validators are extensions of DataValidation, this will not delve into how the validators can be assembled, especially the composite validators, since the Data Validation component already does this. Instead, this demo section will simply show several scenarios and which validators would be applicable to handle them.

4.3.3.1 Team Captain and Free Agent

This scenario has the following conditions (with assumed IDs for the various aspects):

- Registering member must not be barred
- If the project is of category “Assembly” (category Id=4), then registering member must not be currently a Team Member (roleId = 3) for this project
- Member registering as Team Captain must not be currently a Team Captain for this project
- Member registering as a Free Agent (roleId = 2) must not be currently a Team Captain with assigned Team

Suppose the registrant has user ID = 1 and is a Free Agent, and the project is of category “Assembly” and ID 5.. The following validators would be used (with appropriate BundleInfo)

```
BundleInfo bundleInfo = new BundleInfo();
bundleInfo.setBundle("mybundle");
bundleInfo.setDefaultMessage("./test_files/myTemplate.txt");

// Suppose the registrant has user ID = 1 and is a Free Agent, and the
// project is of category  “Assembly” and ID 5.
RegistrationInfo info = new RegistrationInfoImpl();
info.setProjectId(5);
info.setRoleId(2);
info.setUserId(1);

// Create FullProjectData for test.
FullProjectData project = createFullProjectDataForTest();

// sets the user information. The user is not registered, has rating
// 900, and the project is in registration
RegistrationInfo info1 = new RegistrationInfoImpl();
info1.setProjectId(2);
info1.setRoleId(3);
info1.setUserId(2);
RatingInfo ratingInfo = new RatingInfo(RatingType.DESIGN, 900, 5, 20);
ExternalUser user = new ExternalUserImpl(1, "handle1", "user1",
        "TCMember", "user1@topcoder.com");
((ExternalUserImpl) user).addRatingInfo(ratingInfo);

// The following validators would be used (with appropriate BundleInfo)

// member is not barred
```

```
        MemberNotBarredValidator val1 = new MemberNotBarredValidator(bundleInfo);

        // if project is in "Assembly" category, the registrant is not yet a
        // team member.
        MemberNotTeamMemberForProjectValidator val21 = new
MemberNotTeamMemberForProjectValidator(
                bundleInfo);
        ProjectCategoryConditionalValidator val2 = new ProjectCategoryConditionalValidator(
                val21, 4);

        // Check that member registering as team captain as is not a team
        // captain on the team already.
        MemberNotRegisteredWithRoleForProjectValidator val3 =
            new MemberNotRegisteredWithRoleForProjectValidator(
                bundleInfo, 3);

        // Check that member is not a team captain with registrants
        MemberNotTeamCaptainWithMembersForProjectValidator val4
            = new MemberNotTeamCaptainWithMembersForProjectValidator(
                bundleInfo, 3);

        // These four validators (val1, val2, val3, and val4) would be packaged
        // into an Array and put in an AndValidator and passed to the
        // DataValidationRegistrationValidator instance during construction.
        AndValidator validator = new AndValidator(new ConfigurableValidator[] {
            val1, val2, val3, val4 }, false);
        ProjectServices projectServices = new MockProjectServicesImpl();
        BanManager banManager = new MockBanManager();
        TeamManager teamManager = new MockTeamManagerImpl();
        DataValidationRegistrationValidator registrationValidator =
            new DataValidationRegistrationValidator(
                teamManager, projectServices, banManager, null, validator);

        OperationResult result3 = registrationValidator.validate(info, user,
                project);
```

4.3.3.2 Tournaments

In case there's a tournament going on, a new custom validator may be developed and added to the above configuration with the following rule:
If the project is of type "Contest" (type Id=5), then a registering member must be registered to the tournament.

Suppose our custom validator is of type MemberMustBeRegisteredValidator. It would be packaged in a ProjectTypeConditionalValidator that would be only activated if the project type is "Contest". The ProjectTypeConditionalValidator instance would then be passed to the DataValidationRegistrationValidator instance during construction.

```
        BundleInfo bundleInfo = new BundleInfo();
        bundleInfo.setBundle("mybundle");
        bundleInfo.setDefaultMessage("./test_files/myTemplate.txt");
        // Creates an instance of the custom validator MemberMustBeRegisteredValidator.
```

```
MemberMustBeRegisteredValidator innerValidator = new MemberMustBeRegisteredValidator(
        bundleInfo);

// Packages the custom validator in a ProjectTypeConditionalValidator that would be
// only activated if the project type is "Contest".
ProjectTypeConditionalValidator validator = new ProjectTypeConditionalValidator(
        innerValidator, 5);

ProjectServices projectServices = new MockProjectServicesImpl();
BanManager banManager = new MockBanManager();
TeamManager teamManager = new MockTeamManagerImpl();
// Passes to the ProjectServices instance to the DataValidationRegistrationValidator
// instance during construction.
DataValidationRegistrationValidator registrationValidator =
    new DataValidationRegistrationValidator(
        teamManager, projectServices, banManager, null, validator);
```

5. Future Enhancements

More validators may be added. Also conditionals based on other properties.