[TopCoder]

Data Validation 1.1 Requirements Specification

1. Scope

1.1 Overview

The Data Validation component analyses objects to verify that the content meets defined rules. Object content is different for each class and is defined by each class specific DataValidation handler. For example, an Integer handler could validate that the object passed in is in fact an integer and falls within a specified range. In addition, data can be validated across object types.

In version 1.1, support for returning validation messages as a collection of Strings is added. This way, the component is more flexible for GUI interfaces to format and display validation error messages.
Also, support for internationalization is required in this new version without loosing backward compatibility.

1.2 Logic Requirements

1.2.1 Data Validation

- Provide a design to facilitate data validation.
- A data validation handler for each supported class defines how the object is to be validated.
- The initial design should provide handlers for each primitive data type and their corresponding object wrapper.
- Objects can be validated against different target class. For example, a string can be validated against the Integer handler to validate that the string can in fact be converted to an integer and falls within a defined range.
- Multiple handlers can be defined for a target class. For example, a company may have numerous handlers that validate numbers. Different handlers may focus on salaries, age and statistics.
- Custom handlers can be defined to validate complex data types.
- The component should support a pluggable interface to enable additional handlers to be registered.
- When performing composite validations, each atomic validation message must be a single element of the validation result message collection. In case the validation does not succeed, the result message list must contain the result of all evaluated unsuccessful validations.
- If a composite validation is itself composed by composite validations forming a validation tree, the message collection must contain the messages of all evaluated unsuccessful leaves.
- For composite validations, the evaluation of sub-elements will stop when the composite has already failed.
- In case one or more validation items of a composite validation does not succeed but the composite as a whole is successful the return message list must be empty (i.e. “OR” composite validation).
- The component must support retrieving messages from resource bundles to support I18N according to Java 1.4 model.
- To provide backward compatibility with 1.0 version, existing interface, class and method names and behavior must not change.

1.3 Required Algorithms

- Must define how handlers are managed by the Data Validation component.

1.4 Example of the Software Usage

The Data Validation component can be used to facilitate server side data validation for web-based applications. The component can be used to validate that input parameters, which are all represented as strings, are of the correct type, length and within a specified range.
In case there are multiple validations errors, the web page can display them as a bullet list. Also, the application will be able to display messages in the language of choice, providing the appropriate resource bundles.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

The `ObjectValidator` Interface must include the following method for retrieving the message collection in the default locale:

```
String[] getMessages(Object obj);
```

In case validation is successful, the return value must be an empty array. In case validation is not successful, it will be not empty.

2.1.3 Environment Requirements

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

2.1.4 Package Structure

com.topcoder.util.datavalidator

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

Data Handlers must be pluggable.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

None.

3.2.2 TopCoder Software Component Dependencies:

None specified.

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

None.

3.2.4 QA Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000

- Windows 2003

3.3 Design Constraints

3.3.1 Development Standards:

- The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon.