|  |  |  |
| --- | --- | --- |
|  | |  |
|  | | Requirements Document |
|  | |  |
|  | Validator subsystem requirements  **Version: 1.0**  **Last Revised:** December 20, 2010  **Author: Anatoliy Lokshin** | |

Table of Contents

[Feature Overview 4](#_Toc277689729)

[1 Terminology 4](#_Toc277689730)

[2 Product Requirements 4](#_Toc277689731)

[2.1 Functional Area 5](#_Toc277689732)

Document Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Author** | **Revision No.** | **Date** | **Description of Change** |
| **Anatoliy Lokshin** | 1.0 | 11/17/2010 | First revision |
| **Anatoliy Lokshin** | 1.1 | 01/06/2011 | Fixed issue SECFRM-247 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Approvers

|  |  |  |
| --- | --- | --- |
| **Name** | **Title** | **Date Approved** |
| **Kyle Quest** | Architect | 11/18/2010 |
| **Julia Kuchmai** | QA Representative | 11/18/2010 |
|  |  |  |

Reviewers

| **Name** | **Title** | **Date Reviewed** |
| --- | --- | --- |
| **Kyle Quest** | Architect | 11/18/2010 |
| **Julia Kuchmai** | QA Representative | 11/18/2010 |
|  |  |  |

References

|  |  |  |
| --- | --- | --- |
| **Document Name** | **Author** | **Location** |
| MetraTech Security Framework Specification | Kyle Quest | http://seceng.metratech.com/gf/download/docmanfileversion/20/78/MtSecurityFrameworkSpec.doc |
|  |  |  |

# Feature Overview

The Validator Subsystem provides data type, data format, data value, and other data property validation services. These data validation services are implemented using a set of configurable validation engines.

# Terminology

Requirements that are labeled with an R (such as R-100) are Mandatory (or Required) for the targeted release.

Requirements that are labeled with an O (such as O-100) are Optional for the targeted release.

It is left to Engineering’s discretion on whether these can be met for the targeted release but it is imperative that Engineering take these into consideration when making any design or infrastructure decisions. This holds good for any requirement that might be listed under “Future Requirements” section as well.

**(R-1)** It is a Mandatory requirement that the infrastructure developed shall support the ‘feasibility’ of all feature requirements listed in the document – though the delivery of the features themselves could be scheduled for a later time.

# Product Requirements

The configuration for the Validator Subsystem controls what validator engines are loaded for different validator engine categories. There can be more than one validator engine instance for any given category as long as the engine ID values are unique. The validator engines can be configured using a set of properties and validation rules (e.g., string patterns, value ranges, length limits, etc).

In addition to confirming one or more data properties validators are also responsible for converting the data parameter into its expected data type format if the data properties pass the validator requirements. For basic data types it means converting the data parameters into integers, strings, etc.

Validator engines can be grouped into the following categories:

* Basic Integer data type validators.
* Basic Long integer data type validators.
* Basic Double data type validators.
* Basic String data type validators.
* Printable String data validators.
* Hexadecimal String data validators.
* Pattern String data validators.
* Date String data validators.
* Credit Card Number String data validators.
* Base 64 encoded data validators.

## Functional Area

| Requirement | Story | Constraints |
| --- | --- | --- |
| R-500 – Mandatory requirement | All validator engines check input value for null or empty string and throw the NullInputDataException exception when it is.  An exception is Basic String Validator. It passes empty string thru while MinLength parameter is set to 0 (see R-506 for details). |  |
| R-501 – Mandatory requirement | All validator engines throw ValidatorInputDataException exception with appropriate message if the validation process was unsuccessful. |  |
| R-502 – Mandatory requirement | Validator subsystem provides a set of configurable engines validating input data. Configuration can be stored in XML configuration files and read by the Configuration loader. Below is a list of validation engines to be implemented (one for each of categories). |  |
| R-503 – Mandatory requirement | Basic Integer data type validator confirms that the input parameter data is an Integer number and it’s within an expected value range. If the validation process is successful it returns an Integer object with the value from the input parameter data, otherwise an exception is thrown.  Parameters:  MinValue – Lower value range bound (optional, default -2,147,483,648)  MaxValue – Upper value range bound (optional, default 2,147,483,647) |  |
| R-504 – Mandatory requirement | Basic Long integer data type validator confirms that the input parameter data is a Long integer number and it’s within an expected value range. If the validation process is successful it returns a Long integer object with the value from the input parameter data, otherwise an exception is thrown.  Parameters:  MinValue – Lower value range bound (optional, default -9,223,372,036,854,775,808)  MaxValue – Upper value range bound (optional, 9,223,372,036,854,775,807) |  |
| R-505 – Mandatory requirement | Basic Double data type validator confirms that the input parameter data is a Double precision number and it’s within an expected value range. If the validation process is successful it returns a Double number object with the value from the input parameter data, otherwise an exception is thrown.  Parameters:  MinValue – Lower value range bound (optional, -1.7e308)  MaxValue – Upper value range bound (optional, 1.7e308) |  |
| R-506 – Mandatory requirement | Basic String data type validator confirms that the input parameter data is a String and it fits within an expected length requirement. Also it confirms that the input string does not contain disabled characters (black list) or vice versa, contains only enabled characters (white list). All leading and trailing spaces can be ignored. If the validation process is successful it returns a String object with the value (**optionally trimmed**) from the input parameter data, otherwise an exception is thrown.  Parameters:  MinLength – minimum string length (optional, default 0)  MaxLength – maximum string lengtn (optional, default 4294967295)  CharSet – a set of characters to validate a string against (optional)  WhiteList – indicates the CharSet has to be treated as allowed characters (optional, default true)  BlackList – indicates the CharSet has to be treated as disabled characters (optional, default false)  TrimInput – indicates to remove any leading and trailing spaces before validation process and return value trimmed (optional, default false) |  |
| R-507 – Mandatory requirement | Printable String data validator confirms that the input parameter data is a String containing only printable ASCII characters (with codes from 32 till 126). All leading and trailing spaces can be ignored. If the validation process is successful it returns a String object with the value (**optionally trimmed**) from the input parameter data, otherwise an exception is thrown.  Parameters:  TrimInput – indicates to remove any leading and trailing spaces before validation process and return value trimmed (optional, default false) |  |
| R-508 – Mandatory requirement | Hexadecimal String data validator confirms that the input parameter data is a String containing only characters used to represent hexadecimal number values. If the validation process is successful it returns a String object with the value from the input parameter data, otherwise an exception is thrown. |  |
| R-509 – Mandatory requirement | Pattern String data validator confirms that the input parameter data is a String conforms to the regular expression pattern active in the engine. If the validation process is successful it returns a String object with the value from the input parameter data, otherwise an exception is thrown.  Parameters:  ExtraParams – contains a list of regular expressions to validate an input against  WhiteList – indicates the ExtraParams has to be treated as allowed characters (optional, default true)  BlackList – indicates the ExtraParams has to be treated as disabled characters (optional, default false)  TrimInput – indicates to remove any leading and trailing spaces before validation process and return value trimmed (optional, default false) |  |
| R-510 – Mandatory requirement | Date String data validator confirms that the input parameter data is a String containing a date. A value can be passed in the following date and time formats (from the current Windows regional settings):   * **Short date [Short time] OR [Long time]** * **Long date [Long time]** * **YYYY-MM-DD[ THH:mm:[ss[.msec]]]**   The validator automatically tries to cast an input value to date and time.  If the validation process is successful it returns a **DateTime** object with the value from the input parameter data, otherwise an exception s thrown. |  |
| R-511 – Mandatory requirement | Credit Card Number String data validator confirms that the input parameter data is a String containing a possibly valid credit card number. The **Luhn algorithm** is used for the verification. CNN consists of 13 to 16 decimal digits. It’s allowed to split CCN digits with spaces or hyphens. If the validation process is successful it returns a String object with the **trimmed value** with digital characters only from the input parameter data, otherwise an exception is thrown. |  |
| R-512 – Mandatory requirement | Base 64 string validator confirms that the input parameter is a String containing Original BASE 64 for PEM encoded data. If the validation process is successful it returns a String object with the value from the input parameter data, otherwise an exception is thrown. |  |