[TopCoder]

File Format Validator 1.0 Component Specification

1. Design

IBM has a Worldwide Pricing Tool Cost which takes from several different sources, provided by different groups and systems. Those files come in different formats with different content.

The application provided in this component allows users to validate their file format prior to uploading into the Pricing Tool. This avoids issues related to processing files that will be rejected in the future by the application due to incorrect formats.

The main advantage of File Format Validation Tool (FFVT) is saving user time and application process time by allowing a way to proactively perform those validations.

The user will notice there is an issue detected by the application prior to uploading it to the Pricing Tool. If that's the case he can take the following corrective actions to solve the issue:

- Work with the information providers and understand whether a change is required along the generating process.
- Notify Pricing Tool team if the file format should be acceptable – request a fix in the application.

This component provides programmatic API (see TextFileFormatValidator class) and a desktop application (see FileFormatValidationTool class) for validating text files. Currently validation of CSV and fixed length field files is supported. It's assumed that each file to be validated can contain lines of 3 different types (header, data and footer) in addition to all lines to be ignored (e.g. comments). Header lines should go at the very beginning of the file (not taking ignored lines into account), and footer lines should go at the very end of the file (again, not taking ignored lines into account). This component can check whether an expected number of lines of each type are provided in the validated file.

Additionally fields from each line can be validated using the following rules:

- field value should be matched with some predefined regular expression pattern;
- field value should represent some data type (integer, big decimal or date/time);
- field should hold a number that is equal/not equal/less than/greater than/less than or equal/greater than or equal to some predefined value;
- field should hold a date/time that is equal/not equal/less than/greater than/less than or equal/greater than or equal to some predefined date/time value;
- field should hold a number that is equal/not equal/less than/greater than/less than or equal/greater than a number from some other field;
- field should hold a date/time that is equal/not equal/less than/greater than/less than or equal/greater than a date/time from some other field.

1.1 Design Patterns

Strategy pattern – TextFileFormatValidator uses pluggable FieldExtractor and FieldsValidator instances; FileFormatValidatorFrame uses pluggable FileFormatValidator and ValidationReportGenerator instances.

Template method pattern – SingleFieldValidator#validate(fieldValues, errors) and TwoFieldsComparisonValidator#validate(fieldValues, errors) are template methods that use abstract namesake overload methods to validate specific field values.

Observer pattern – ChooseFileButtonListener's actionPerformed() method is called by Swing to notify that a button was pressed.

1.2 Industry Standards

Swing, XML, DOM, XSD, CSV

1.3 Required Algorithms

1.3.1 Logging

This component must perform logging in all public methods of PlainTextValidationReportGenerator, TextFileFormatValidator, CSVFieldExtractor, FixedLengthFieldExtractor, SingleFieldValidator and FileFormatValidationTool and additionally in private FileFormatValidatorFrame#performValidation() method.

All information mentioned below must be logged using log:Log or LOG:Log attribute.

In all mentioned methods method entrance with input arguments, method exit with call duration time and returned values must be logged at DEBUG level.

All errors and all caught exceptions must be logged at ERROR level.

1.3.2 XSD for configuration file validation

The content of XSD file to be used for validating XML configuration files for TextFileFormatValidator depends a lot on what exact FieldExtractor and FieldsValidator implementations will be used together with TextFileFormatValidator. The XSD file provided below supports all FieldExtractor and FieldsValidator implementations provided in this component. In future if some new implementation is added, the XSD file should be updated respectively.

config.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           attributeFormDefault="unqualified" elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="lineTypeName">
    <xs:restriction base="xs:string">
      <xs:enumeration value="header"/>
      <xs:enumeration value="data"/>
      <xs:enumeration value="footer"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="fieldTypeName">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Integer"/>
      <xs:enumeration value="BigDecimal"/>
      <xs:enumeration value="Date"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="operatorName">
    <xs:restriction base="xs:string">
      <xs:enumeration value="equal"/>
      <xs:enumeration value="notEqual"/>
      <xs:enumeration value="lessThan"/>
      <xs:enumeration value="greaterThan"/>
      <xs:enumeration value="lessOrEqual"/>
      <xs:enumeration value="greaterOrEqual"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="yesNo">
    <xs:restriction base="xs:string">
      <xs:enumeration value="yes"/>
      <xs:enumeration value="no"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="config">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" maxOccurs="3" name="lineType">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="extractor" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element minOccurs="0" maxOccurs="unbounded" name="field">
```

```
                    <xs:complexType>
                      <xs:attribute name="startPos" type="xs:positiveInteger"
                                    use="required" />
                      <xs:attribute name="endPos" type="xs:positiveInteger"
                                    use="required" />
                      <xs:attribute name="trim" type="yesNo" use="optional"
                                    default="yes" />
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="type" type="xs:string" use="required" />
                <xs:attribute name="separator" type="xs:string" use="optional" />
              </xs:complexType>
            </xs:element>
            <xs:element name="validators" minOccurs="0" maxOccurs="1">
              <xs:complexType>
                <xs:sequence>
                  <xs:element minOccurs="1" maxOccurs="unbounded" name="validator">
                    <xs:complexType>
                      <xs:attribute name="type" type="xs:string" use="required" />
                      <xs:attribute name="fieldIndex" type="xs:positiveInteger"
                                    use="optional" />
                      <xs:attribute name="pattern" type="xs:string" use="optional" />
                      <xs:attribute name="operator" type="operatorName" use="optional" />
                      <xs:attribute name="value" type="xs:string" use="optional" />
                      <xs:attribute name="fieldType" type="fieldTypeName" use="optional" />
                      <xs:attribute name="format" type="xs:string" use="optional" />
                      <xs:attribute name="skipIfEmpty" type="yesNo" use="optional"
                                    default="no" />
                      <xs:attribute name="fieldIndex1" type="xs:positiveInteger"
                                    use="optional" />
                      <xs:attribute name="fieldIndex2" type="xs:positiveInteger"
                                    use="optional" />
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="type" type="lineTypeName" use="required" />
          <xs:attribute name="pattern" type="xs:string" use="optional" />
          <xs:attribute name="num" type="xs:positiveInteger" use="optional" />
          <xs:attribute name="minNum" type="xs:nonNegativeInteger" use="optional" />
          <xs:attribute name="maxNum" type="xs:positiveInteger" use="optional" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="formatName" type="xs:string" use="required" />
    <xs:attribute name="ignoredLinePattern" type="xs:string" use="optional" />
  </xs:complexType>
 </xs:element>
</xs:schema>
```

1.4 Component Class Overview

CSVFieldExtractor

This class is an implementation of FieldExtractor that can extract field values from CSV rows. This class supports not only a comma as a separator, but any separator matched with the configurable regexp pattern. This class uses Apache Commons Logging library to perform logging of errors and debug information.

ChooseFileButtonListener

This class is an implementation of ActionListener that is used for performing an action when choose file button is pressed.

ComparisonType [enum]

This is an enumeration for comparison types supported by SingleFieldComparisonValidator and TwoFieldsComparisonValidator subclasses.

DateComparisonValidator

This is a base class for all single field validators that compare a date/time field value with some predefined date/time constant. This class extends SingleFieldComparisonValidator that holds the comparison type to be used.

FieldExtractor [interface]

This interface represents a field extractor to be used by TextFileFormatValidator. It defines a method for extracting field values from a text line. Implementations of this interface must provide a constructor that accepts only configuration DOM Element instance to be compatible with TextFileFormatValidator.

FieldParams

This is an inner class of FixedLengthFieldExtractor that is a container for information about a single field of the file that has fixed length field format. It is a simple JavaBean (POJO) that provides getters and setters for all private attributes and performs no argument validation in the setters.

FieldType [enum]

This is an enumeration for field types supported by FieldTypeValidator.

FieldTypeValidator

This class is an implementation of FieldsValidator that checks whether a field value can be converted to a predefined type.

FieldsValidator [interface]

This interface represents a fields validator. It defines a method for validating the specified row field values. Implementations of this interface must provide a constructor that accepts only configuration DOM Element instance to be compatible with TextFileFormatValidator.

FileFormatValidationTool

This is main class of desktop GUI application that performs file format validation. This class simply reads configuration Properties file and delegates execution to FileFormatValidatorFrame. This class uses Apache Commons Logging library to perform logging of errors and debug information.

FileFormatValidator [interface]

This interface represents a file format validator. It defines a method for validating a file identified with a file path. Implementations of this interface must provide a constructor that accepts only configuration file path to be compatible with FileFormatValidationTool.

FileFormatValidatorFrame

This class represents the main frame of the file format validation tool GUI application. This frame contains 3 text fields for entering input, config and output file paths, 3 buttons for choosing files from the disk using JFileChooser, "Validate" button and text area for validation result report. This class also contains logic for validating configuration files using predefined XSD file and writing validation report to the file. To perform the actual file format validation this class uses a configurable FileFormatValidator implementation instance. To generate a validation report this class uses a configuration ValidationReportGenerator implementation instance. This class uses Apache Commons Logging library to perform logging of errors and debug information.

FixedLengthFieldExtractor

This class is an implementation of FieldExtractor that can extract field values from files that have fixed length field format. This class can optionally trim each retrieved field value. This class uses Apache Commons Logging library to perform logging of errors and debug information.

LineType [enum]

This is an enumeration for line types that are supported by TextFileFormatValidator.

LineTypeDetails

This is an inner class of TextFileFormatValidator that is a container for configuration parameters associated with a single line type. It is a simple JavaBean (POJO) that provides getters and setters for all private attributes and performs no argument validation in the setters.

NumberComparisonValidator
This is a base class for all single field validators that compare a numeric field value with some predefined number. This class extends SingleFieldComparisonValidator that holds the comparison type to be used.

PlainTextValidationReportGenerator
This class is an implementation of ValidationReportGenerator that generates plain text reports for the given ValidationResult instances using the configurable report template.

RegexFieldValidator
This class is an implementation of FieldsValidator that checks whether a field value matches some predefined regular expression pattern.

SingleFieldComparisonValidator [abstract]
This is a base class for all single field validators that compare a field value with some predefined constant.

SingleFieldValidator [abstract]
This class is an abstract implementation of FieldsValidator that is a base class for all validators that can validate a single field with configured index only. This class uses Apache Commons Logging library to perform logging of errors and debug information.

TextFileFormatValidator
This class is an implementation of FileFormatValidator that can be used for validating text files that contain tabular data. It assumes that each line of a file contains data for a single row and supports three different line types: header, data and footer. Lines of each type can have different formats. This class uses pluggable FieldExtractor instances to extract field values from the text lines, and then it uses pluggable FieldsValidator instance to validate extracted field values. This class uses Apache Commons Logging library to perform logging of errors and debug information.

TwoDateFieldsComparisonValidator
This class is an implementation of FieldsValidator that compares two date/time field values to each other when validating them.

TwoFieldsComparisonValidator [abstract]
This class is an implementation of FieldsValidator that validates two fields at predefined positions by comparing them to each other using the configured comparison type. This class uses Apache Commons Logging library to perform logging of errors and debug information.

TwoNumberFieldsComparisonValidator
This class is an implementation of FieldsValidator that compares two numeric field values to each other when validating them.

ValidationContext
This class holds information that is specific to a file type validation context. Separate ValidationContext instance is created and used by TextFileFormatValidator for each validated file.

ValidationReportGenerator [interface]
This interface represents a file type validation report generator. It defines a method for generating a text report for the given ValidationResult instance. Implementations of this interface must provide a constructor that accepts only Properties instance with configuration parameters to be compatible with FileFormatValidationTool.

ValidationResult

This class is a container for file format validation result data. It is a simple JavaBean (POJO) that provides getters and setters for all private attributes and performs no argument validation in the setters.

1.5 Component Exception Definitions

FieldExtractionException

This exception is thrown by implementations of FieldExtractor when some error occurs while extracting the field values (e.g. line has invalid format).

FileFormatValidationException

This exception is thrown by implementations of FileFormatValidator when some fatal error occurs while performing the validation (exception is not thrown when file has invalid format). Also this exception is used as a base class for other specific custom exceptions.

FileFormatValidatorConfigurationException

This runtime exception is thrown by most of classes defined in this component when some error occurs while initializing the class instance using the provided configuration.

ValidationReportGenerationException

This exception is thrown by implementations of ValidationReportGenerator when some error occurs while generating a report for file type validation result.

1.6 Thread Safety

This component is thread safe except its GUI part assuming that internal logging implementation used by Apache Commons Logging library is thread safe.

Implementations of FieldExtractor and FileFormatValidator must be thread safe.

Implementations of ValidationReportGenerator must be thread safe assuming that the provided streams and entities are used by the caller in thread safe manner.

TextFileFormatValidator is immutable and thread safe. It uses thread safe FieldExtractor and FieldsValidator instances.

CSVFieldExtractor and FixedLengthFieldExtractor are immutable and thread safe.

SingleFieldValidator and TwoFieldsComparisonValidator are immutable and thread safe when provided collections are used by the caller in thread safe manner.

ValidationContext is mutable and not thread safe. But it is always used by TextFileFormatValidator in thread safe manner.

LineType, ComparisonType and FieldType are immutable and thread safe enumerations.

LineTypeDetails, FieldParams and ValidationResult are mutable and not thread safe entities.

Implementations of FieldsValidator are required to be thread safe when collections passed to them are used by the caller in thread safe manner.

PlainTextValidationReportGenerator is immutable and thread safe assuming that the provided streams and entities are used by the caller in thread safe manner.

RegexFieldValidator, TwoNumberFieldsComparisonValidator, FieldTypeValidator, NumberComparisonValidator, DateComparisonValidator, TwoDateFieldsComparisonValidator and SingleFieldComparisonValidator are immutable and thread safe when provided collections are used by the caller in thread safe manner.

FileFormatValidationTool is immutable and thread safe. It's safe to use multiple instances of this application when different output file paths are used to avoid file access conflicts.

FileFormatValidatorFrame is not thread safe since its base class is not thread safe.

ChooseFileButtonListener is immutable, but not thread safe since it uses not thread safe JTextField instance. ChooseFileButtonListener should be accessed from the GUI thread.

2. Environment Requirements

2.1 Environment

Development language: Java 1.5
Compile target: Java 1.5
QA Environment: Linux, Windows XP/Vista/7

2.2 TopCoder Software Components

None

2.3 Third Party Components

Apache Commons Logging 1.1.1 (http://commons.apache.org/logging/)

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.ibm.tools.ffvt
com.ibm.tools.ffvt.generators
com.ibm.tools.ffvt.gui
com.ibm.tools.ffvt.text
com.ibm.tools.ffvt.text.extractors
com.ibm.tools.ffvt.text.validators

3.2 Configuration Parameters

3.2.1 Configuration of FileFormatValidationTool and FileFormatValidatorFrame

FileFormatValidationTool loads configuration from the properties file and then passes the created Properties instance to the constructor of FileFormatValidatorFrame. The following table describes the parameters supported in this properties file (unknown parameters are simply ignored):

Parameter	Description	Values
fileFormatValidatorClassName	The full class name of FileFormatValidator instance to be used by the tool. Default is “com.ibm.tools.ffvt.text.TextFileFormatValidator”.	String. Not empty. Optional.
validationReportGeneratorClassName	The full class name of ValidationReportGenerator instance to be used by the tool. Default is “com.ibm.tools.ffvt.generators.PlainTextValidationReportGenerator”.	String. Not empty. Optional.
validationReportGenerator.xxx	Parameters passed to ValidationReportGenerator implementation constructor in a Properties container. Here “xxx” is any string that represents a key in a Properties container passed to ValidationReportGenerator implementation constructor.	String. Multiple. Optional.

3.2.2 *Configuration of PlainTextValidationReportGenerator*

PlainTextValidationReportGenerator loads configuration from the Properties instance passed to its constructor. The following table describes the parameters supported by PlainTextValidationReportGenerator (unknown parameters are simply ignored):

Parameter	Description	Values
reportTemplateFilePath	The path of the text file that holds validation report template. This template can contain the following template fields: %FILE_PATH%, %STATUS% (replaced with “failed” or “succeeded”), %TIMESTAMP%, %FORMAT_NAME% and %ERRORS% (replaced with line break separated list of error descriptions).	String. Not empty. Required.

3.2.3 *Configuration of TextFileFormatValidator*

TextFileFormatValidator loads its configuration parameters from an XML file. The table provided below describes the root document element of this XML file (angle brackets are used for identifying XML elements; parameters without angle brackets are XML attributes):

Parameter	Description	Values
<config>	The root configuration element.	Required.
<config>.formatName	The user-defined format name to be associated with this configuration file. This format name will be specified in the validation result.	String. Not empty. Required.
<config>.ignoredLinePattern	The regular expression pattern that matches all lines those need to be ignored (e.g. comment lines). If not specified, no lines are ignored.	String. Not empty. Optional.
<config>.<lineType>	The configuration for one of line types supported by the validator. If lines of specific type should not appear in the file (e.g. footer), <lineType> element for this type should not be provided.	Multiple. At least one must be provided.
<config>.<lineType>.type	The line type. Must be unique in the configuration file (i.e. no two <lineType> elements with the same “type” attribute can be provided).	String. Required. Must be one of “header”, “data” and “footer” (case insensitive.
<config>.<lineType>.pattern	The regular expression pattern that identifies lines of this type. If not specified, lines will be identified by position in the file and expected lines number.	String. Not empty. Optional.
<config>.<lineType>.num	The exact number of lines of this type that have to be present in the validated file.	String representation of positive integer. Optional.

<config>.<lineType>.minNum	The minimum number of lines of this type that have to be present in the validated file. Is ignored if “num” parameter is specified.	String representation of not negative integer. Optional.
<config>.<lineType>.maxNum	The maximum number of lines of this type that have to be present in the validated file. Is ignored if “num” parameter is specified.	String representation of positive integer. Optional.
<config>.<lineType>. <extractor>	The configuration for field extractor to be used for this line type. This element is passed to the constructor of FieldExtractor implementation.	Required.
<config>.<lineType>. <extractor>.type	The full class name of FieldExtractor implementation to be used for this line type. The following short predefined values are supported: “csv” – for CSVFieldExtractor; “fixed” – for FixedLengthFieldExtractor.	String. Not empty. Required.
<config>.<lineType>. <validators>	The configuration for validators used for validating lines of this type.	Optional.
<config>.<lineType>. <validators>.<validator>	The configuration for one of validators used for validating lines of this type. This element is passed to the constructor of FieldsValidator implementation.	Multiple. Optional.
<config>.<lineType>. <validators>.<validator>.type	The full class name of FieldsValidator to be used for validation. The following short predefined values are supported: “regex” – for RegexFieldValidator; “fieldType” – for FieldTypeValidator; “numberComparison” – for NumberComparisonValidator; “dateComparison” – for DateComparisonValidator; “twoNumberFieldsComparison” – for TwoNumberFieldsComparisonValidator; “twoDateFieldsComparison” – for TwoDateFieldsComparisonValidator.	String. Not empty. Required.

3.2.4 *Configuration of CSVFieldExtractor*

CSVFieldExtractor loads its configuration parameters from an XML element provided to its constructor. The table provided below describes the content this element (angle brackets are used for identifying XML elements; parameters without angle brackets are XML attributes):

Parameter	**Description**	**Values**
separator	The regular expression for separator that separates field values in a line. Default is “,”.	String. Not empty. Optional.

3.2.5 *Configuration of FixedLengthFieldExtractor*

FixedLengthFieldExtractor loads its configuration parameters from an XML element provided to its constructor. The table provided below describes the content this element (angle brackets are used for identifying XML elements; parameters without angle brackets are XML attributes):

Parameter	Description	Values
<field>	The configuration for each field to be extracted. The order of XML elements must correspond to the order of fields in a line.	Multiple. At least one is required.
<field>.startPos	The start position of the field in a line (1-based index).	String representation of positive integer. Required.
<field>.endPos	The end position of the field in a line (1-based, inclusive). Must be greater or equal to “startPos”.	String representation of positive integer. Required.
<field>.trim	The value indicating whether trimming of the field value must be performed. Default is “yes”.	“yes” or “no” (case insensitive. Optional.

3.2.6 *Configuration of SingleFieldValidator subclasses*

SingleFieldValidator and its subclasses load configuration parameters from an XML element provided to its constructor. The table provided below describes the content this element (angle brackets are used for identifying XML elements; parameters without angle brackets are XML attributes; note that SingleFieldValidator subclasses can additionally support other parameters):

Parameter	Description	Values
fieldIndex	The 1-based index of the field to be validated.	String representation of positive integer. Required.
skipIfEmpty	The value indicating whether field validation must be skipped in case if the field value is empty. Default is “no”.	“yes” or “no” (case insensitive. Optional.

3.2.7 *Configuration of RegexFieldValidator*

RegexFieldValidator loads its configuration parameters from an XML element provided to its constructor. The table provided below describes the content this element (angle brackets are used for identifying XML elements; parameters without angle brackets are XML attributes):

Parameter	Description	Values
pattern	The regular expression pattern to be used for checking field values.	String. Not empty. Required.

Also see section 3.2.6 for additional parameters.

3.2.8 *Configuration of FieldTypeValidator*

FieldTypeValidator loads its configuration parameters from an XML element provided to its constructor. The table provided below describes the content this element (angle brackets are used for identifying XML elements; parameters without angle brackets are XML attributes):

Parameter	Description	Values

fieldType	The expected field type.	“Integer”, “BigDecimal” or “Date” (case insensitive. Required.
format	The expected format of the field type. Currently supported for “Date” field type only (see docs of class SimpleDateFormat in JDK for details).	String. Not empty. Required for “Date”, ignored for other field types.

Also see section 3.2.6 for additional parameters.

3.2.9 *Configuration of SingleFieldComparisonValidator subclasses*

SingleFieldComparisonValidator and its subclasses load configuration parameters from an XML element provided to its constructor. The table provided below describes the content this element (angle brackets are used for identifying XML elements; parameters without angle brackets are XML attributes; note that SingleFieldComparisonValidator subclasses can additionally support other parameters):

Parameter	Description	Values
operator	The operator that describes the comparison type to be used.	“equal”, “notEqual”, “lessThan”, “qreaterThan”, “lessOrEqual” or “greaterOrEqual” (case insensitive). Required.

Also see section 3.2.6 for additional parameters.

3.2.10 *Configuration of NumberComparisonValidator*

NumberComparisonValidator loads its configuration parameters from an XML element provided to its constructor. The table provided below describes the content this element (angle brackets are used for identifying XML elements; parameters without angle brackets are XML attributes):

Parameter	Description	Values
value	The constant value to be used for comparison.	String representation of BigDecimal. Required.

Also see sections 3.2.6 and 3.2.9 for additional parameters.

3.2.11 *Configuration of DateComparisonValidator*

DateComparisonValidator loads its configuration parameters from an XML element provided to its constructor. The table provided below describes the content this element (angle brackets are used for identifying XML elements; parameters without angle brackets are XML attributes):

Parameter	Description	Values
format	The date/time format to be used for parsing field value and “value” parameter (see docs of class SimpleDateFormat in JDK for details).	String. Not empty. Required.
value	The constant date/time value to be used for comparison. The format of this value must match “format” parameter.	String. Not empty. Required.

Also see sections 3.2.6 and 3.2.9 for additional parameters.

3.2.12 *Configuration of TwoFieldsComparisonValidator subclasses*

TwoFieldsComparisonValidator and its subclasses load configuration parameters from an XML element provided to its constructor. The table provided below describes the content this element (angle brackets are used for identifying XML elements; parameters without angle brackets are XML attributes; note that TwoFieldsComparisonValidator subclasses can additionally support other parameters):

Parameter	Description	Values
fieldIndex1	The 1-based index of the first field to be validated (is used as the first operand during the comparison).	String representation of positive integer. Required.
fieldIndex2	The 1-based index of the second field to be validated (is used as the second operand during the comparison).	String representation of positive integer. Required.
skipIfEmpty	The value indicating whether field validation must be skipped in case if any of two field values is empty. Default is “no”.	“yes” or “no” (case insensitive. Optional.
operator	The operator that describes the comparison type to be used.	“equal”, “notEqual”, “lessThan”, “qreaterThan”, “lessOrEqual” or “greaterOrEqual” (case insensitive). Required.

3.2.13 *Configuration of TwoDateFieldsComparisonValidator*

TwoDateFieldsComparisonValidator loads its configuration parameters from an XML element provided to its constructor. The table provided below describes the content this element (angle brackets are used for identifying XML elements; parameters without angle brackets are XML attributes):

Parameter	Description	Values
format	The date/time format to be used for parsing field values (see docs of class SimpleDateFormat in JDK for details).	String. Not empty. Required.

Also see section 3.2.12 for additional parameters.

3.3 Dependencies Configuration

Apache Commons Logging Wrapper should be configured to use Log4j logger.

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

Please see the demo.

4.3 Demo

4.3.1 *API usage*

```
// Create an instance of TextFileFormatValidator
```

```
FileFormatValidator fileFormatValidator = new TextFileFormatValidator("csv_config.xml");

// Validate a file
ValidationResult validationResult = fileFormatValidator.validate("sample.csv");

// Check if validation was successful
boolean successful = validationResult.isSuccessful();

// Print out validation errors if not successful
if (!successful) {
    for (String error : validationResult.getErrors()) {
        System.out.println(error);
    }
}
```

4.3.2 *Usage of GUI application*

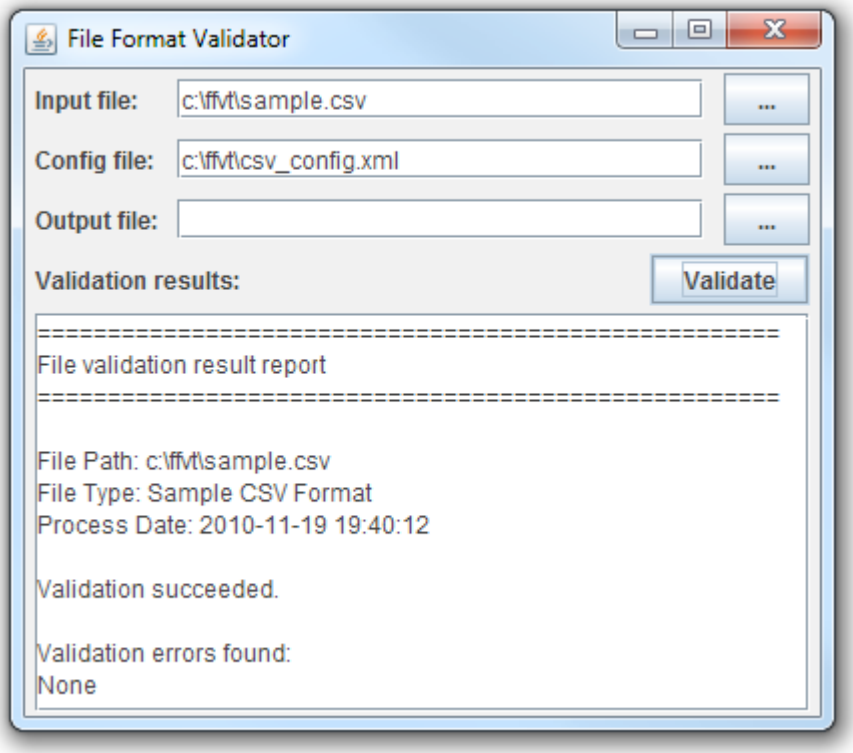
The following command line can be used for starting a GUI application when FileFormatValidationTool.properties configuration file exists:

```
java -jar FileFormatValidationTool.jar
```

If the application need to be executed using a custom configuration file, the following command line can be used:

```
java -jar FileFormatValidationTool.jar custom_config.properties
```

The image provided below shows how the working GUI application should look like:



To perform the file format validation using the GUI application the user must follow the next steps:

- Enter path of the file to be validated in “Input file” text field (or use the corresponding “...” button to select a file on the disk using the dialog).
- Enter path of the configuration file to be used for validation in “Config file” text field (or use the corresponding “...” button to select a file on the disk using the dialog). This step is optional since the application can read configuration file name from properties.
- Optionally enter path of the file to which validation result report must be saved in “Output file” text field (or use the corresponding “...” button to select a file on the disk using the dialog).

- Click “Validate” button.
- See validation result report in the text area and optionally in the output file.

4.3.3 *Sample FileFormatValidationTool configuration file*

FileFormatValidationTool.properties

```
fileFormatValidatorClassName=com.ibm.tools.ffvt.text.TextFileFormatValidator
validationReportGeneratorClassName=\
    com.ibm.tools.ffvt.generators.PlainTextValidationReportGenerator
validationReportGenerator.reportTemplateFilePath=report.txt
```

report.txt

```
=====================================================
File validation result report
=====================================================

File Path: %FILE_PATH%
File Type: %FORMAT_NAME%
Process Date: %TIMESTAMP%

Validation %STATUS%.

Validation errors found:
%ERRORS%
```

4.3.4 *Sample validation of file with fixed length field format*

Assume that the following configuration file is used:

fixed_config.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="config.xsd"
        formatName="Sample Fixed Column Length Format"
        ignoredLinePattern="\*.*">
  <lineType type="header" pattern="H .*" num="1">
    <extractor type="fixed">
      <field startPos="3" endPos="7" trim="no"/>
      <field startPos="9" endPos="9"/>
      <field startPos="11" endPos="13"/>
      <field startPos="15" endPos="33"/>
      <field startPos="35" endPos="37"/>
    </extractor>
    <validators>
      <validator type="regex" fieldIndex="1" pattern="[0-9]{5}"/>
      <validator type="regex" fieldIndex="2" pattern="[D|N]"/>
      <validator type="numberComparison" fieldIndex="3"
                 operator="greaterOrEqual" value="123"/>
      <validator type="fieldType" fieldIndex="4" fieldType="Date"
                 format="yyyy-MM-dd-HH:mm:ss"/>
      <validator type="regex" fieldIndex="5" pattern="[A-Z]{3}"/>
    </validators>
  </lineType>
  <lineType type="data" pattern="D .*" minNum="1">
    <extractor type="fixed">
      <field startPos="3" endPos="6"/>
      <field startPos="8" endPos="11"/>
      <field startPos="16" endPos="19"/>
      <field startPos="24" endPos="27"/>
      <field startPos="29" endPos="29"/>
      <field startPos="31" endPos="42"/>
      <field startPos="44" endPos="53"/>
      <field startPos="55" endPos="57"/>
      <field startPos="59" endPos="61"/>
    </extractor>
    <validators>
      <validator type="fieldType" fieldIndex="1" fieldType="Integer"/>
      <validator type="fieldType" fieldIndex="2" fieldType="Integer"
                 skipIfEmpty="yes"/>
      <validator type="fieldType" fieldIndex="3" fieldType="Integer"/>
```

```
      <validator type="fieldType" fieldIndex="4" fieldType="Integer"
                 skipIfEmpty="yes"/>
      <validator type="regex" fieldIndex="5" pattern="[F|U]"/>
      <validator type="regex" fieldIndex="6" pattern="[0-9]{9}\.[0-9]{2}"/>
      <validator type="dateComparison" fieldIndex="7" operator="greaterThan"
                 value="2009-01-01" format="yyyy-MM-dd"/>
      <validator type="twoNumberFieldsComparison" fieldIndex1="9" fieldIndex2="8"
                 operator="greaterThan"/>
    </validators>
  </lineType>
  <lineType type="footer" pattern="T .*" num="1">
    <extractor type="fixed">
      <field startPos="3" endPos="7"/>
      <field startPos="9" endPos="16"/>
      <field startPos="18" endPos="32"/>
    </extractor>
    <validators>
      <validator type="regex" fieldIndex="1" pattern="[0-9]{5}"/>
      <validator type="regex" fieldIndex="2" pattern="[0-9]{8}"/>
      <validator type="regex" fieldIndex="3" pattern="[0-9]{12}\.[0-9]{2}"/>
    </validators>
  </lineType>
</config>
```

And assume that the following file is validated:

fixed_sample.txt

```
H 01413 D 631 2009-10-15-11:09:36 USD
D 4611         1085         F 000000020.38 2009-10-31 110 120
D 9119         9440         F 000000000.00 2010-06-30
D 9119         9441         F 000000000.00 2010-06-30 110 120
D 9119         9442         F 000000000.00 2010-06-30
D 9119         9443         F 000000000.00 2010-06-30 110 120
D 2094 2094    7631    7641 U 000006947.00 2010-06-30
D 2094 2094    7632    7642 U 000003768.00 2010-06-30 110 120
D 2094 2094    7633    7643 U 000003768.00 2010-06-30
T 01413 00000029 000000233500.38
```

Then the following validation report is expected:

```
=======================================================
File validation result report
=======================================================

File Path: c:\ffvt\fixed_sample.txt
File Type: Sample Fixed Column Length Format
Process Date: 2010-11-19 19:48:29

Validation succeeded.

Validation errors found:
None
```

Now assume that input file was modified a little (modifications are marked with red):

fixed_sample.txt

```
H 01413 D 631 2009-10-15-11:09:36 USD
D 4611         1085         F 0000000020.3 2009-10-31 110 120
D 9119         9440         F 000000000.00 2010-06-30
D 9119         9441         F 000000000.00 2010-06-30 110 120
D 9119         9442         F 000000000.00 2008-06-30
D 9119         9443         F 000000000.00 2010-06-30 110 120
D 2094 2094    7631    7641 U 000006947.00 2010-06-30
D 2094 2094    7632    7642 U 000003768.00 2010-06-30 120 110
D 2094 2094    7633    7643 U 000003768.00 2010-06-30
K 01413 00000029 000000233500.38
```

Then the following validation report is expected:

```
=======================================================
File validation result report
=======================================================
```

```
File Path: c:\ffvt\fixed_sample.txt
File Type: Sample Fixed Column Length Format
Process Date: 2010-11-19 19:49:52

Validation failed.

Validation errors found:
Line 2 - field value at position 6 (0000000020.3) is not matched with regexp pattern ([0-
9]{9}\.[0-9]{2})
Line 5 - field value at position 7 must be greater than 2009-01-01
Line 8 - field value at position 9 (110) must be greater than field value at position 8 (120)
Line 10 - type of this line cannot be identified
Not enough lines of footer type were found (minimum expected number is 1)
```

4.3.5 *Sample validation of CSV file*

Assume that the following configuration file is used:

csv_config.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="config.xsd"
        formatName="Sample CSV Format"
        ignoredLinePattern="#.*">
  <lineType type="header" num="1">
    <extractor type="csv" />
    <validators>
      <validator type="regex" fieldIndex="1" pattern="ID"/>
      <validator type="regex" fieldIndex="2" pattern="START"/>
      <validator type="regex" fieldIndex="3" pattern="END"/>
      <validator type="regex" fieldIndex="4" pattern="CODE"/>
    </validators>
  </lineType>
  <lineType type="data" minNum="1">
    <extractor type="csv" />
    <validators>
      <validator type="fieldType" fieldIndex="1" fieldType="Integer"/>
      <validator type="numberComparison" fieldIndex="1"
                 operator="greaterThan" value="0"/>
      <validator type="fieldType" fieldIndex="2" fieldType="Integer"/>
      <validator type="fieldType" fieldIndex="3" fieldType="Integer"/>
      <validator type="twoNumberFieldsComparison" fieldIndex1="3" fieldIndex2="2"
                 operator="greaterOrEqual"/>
      <validator type="regex" fieldIndex="4" pattern="[A-Z]{3}[0-9]{3}"/>
    </validators>
  </lineType>
</config>
```

And assume that the following file is validated:

sample.csv

```
# This is a sample CSV file
ID,START,END,CODE
1,1,5,NBK245
2,6,14,OPL082
3,15,15,APE581
```

Then the following validation report is expected:

```
=======================================================
File validation result report
=======================================================

File Path: c:\ffvt\sample.csv
File Type: Sample CSV Format
Process Date: 2010-11-19 19:52:56

Validation succeeded.

Validation errors found:
None
```

5. Future Enhancements

- Other implementations of FileFormatValidator can be provided to support validation of non-text file types.
- Other implementations of FieldsValidator can be provided.