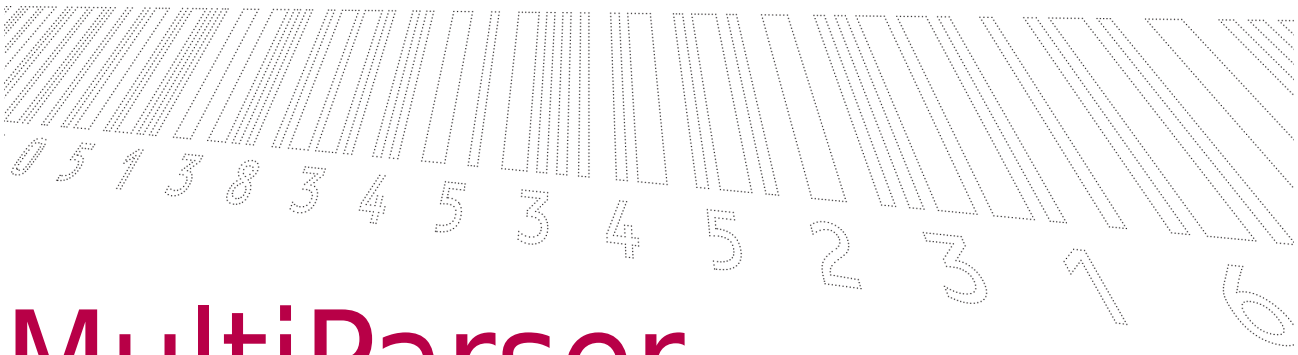
DYNAMIC SOFTWARE

Overview of Functions



MultiParser

For Automotive Manufacturers and Suppliers

Version 1.3.05
Created 05/06/2008

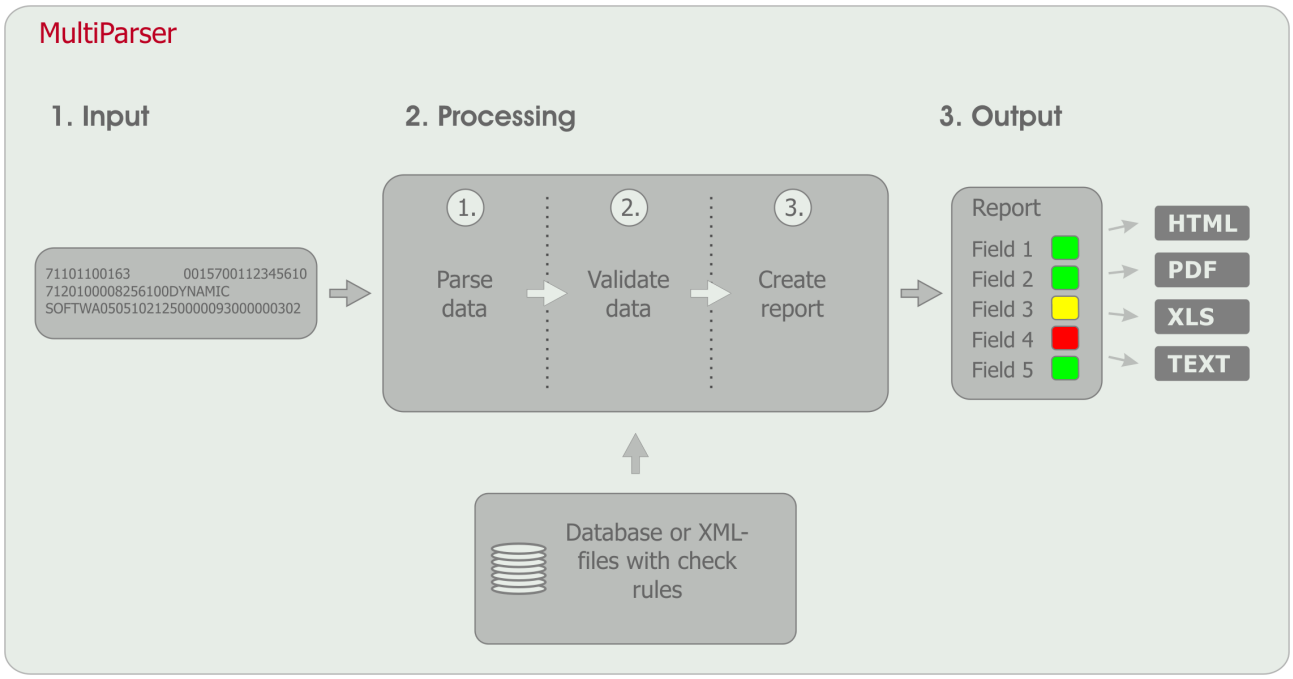
Contents

1 Functions

MultiParser is a generic tool for the analysis of text files that use the standard formats VDA, ODETTE, EDIFACT, or iDOC, which are used in electronic data interchange (EDI).

All check configurations are stored in a configuration database or as a XML file. They can be modified during runtime without any programming. Any number of check configurations can be stored (e.g. ASNs, delivery schedules, credit notes).

Procedure:



1.1 Input

The text to be analysed needs to be an ASCII text file. It can be directly passed to MultiParser as a parameter via a method.

MultiParser can process record-based texts the contents of which are defined by a record key and separated

- either by separators, which is the case with variable character length,
- or by line breaks, which is the case with fixed character length (blocked texts).

1.2 Processing

A text can only be checked if a configuration is available in a database or as a XML file. When an analysis is started a configuration needs to be passed to MultiParser.

A text analysis is based on three steps:

1. Recognise and record data
2. Check data
3. Create report

1.2.1 Parse data

This is the first step of the text analysis. The text to be analysed is read and dissected into a list of text lines. While the text is being read the end of each line is recognised – independently from which operating system (Windows or Unix/Linux) is used. Redundant control characters, such as CarriageReturn, are deleted. However, users can define separators of their own (e.g. "|") in the configuration.

On the basis of the configuration the text lines are dissected into individual fields. For example, the configuration may include the record type 'XYZ' which contains the customer number field starting at the 5th position and ending at the 10th position. MultiParser reads the text from the 5th to the 10th position and assigns the value it read to the customer number field. In the next step the value is validated.

1.2.2 Validate data

The second step forms the core of the MultiParser. In this step the data is validated. The validation is based on a configuration which is either stored in a database or available as a XML file (see chapter 2.2). The validation check is done sequentially, record by record, field by field. For each field exists an individual list of checks. If one of the specifications is not fulfilled an error or at least a warning will be reported for this field in a report of the check results. In this report, an error overwrites a warning.

1.2.3 Create report

All validation check results are administered in a MultiParser-internal report. This report is only built up in the program memory and thus does not have a known document format. This has the advantage that no format is defined for the rendering of the report, which means that the rendering is not restricted to a certain format. The system-internal report is not persistent. It will be reset when a new validation check is run.

1.3 Output

The system-internal report contains the validation check results for each field. Depending on the customer's requirements and environment, a feature for the visualisation of the report can be implemented. MultiParser includes by default a feature allowing text output in a simple format. Without much effort, the report can also be output in the CSV or HTML format. HTML generation can be done on the fly in a servlet or in a JSP page.

By the use of external Java libraries, the system-internal report can also be converted effortlessly into the following formats:

- PDF
- XLS
- HTML
- TEXT

2 Validating rules

2.1 Standard validation checks

Name	Field check description
Min	Defines the minimum value for a field. Example: The unloading point must be greater or equal to 100.
Max	Defines the maximum value for a field. Example: The gross weight must be smaller or equal to 1.000.
Min. digits after comma	Defines the minimum number of digits to the right of the comma. Example: The price must have 2 digits after the comma.
Max. digits after comma	Defines the maximum number of digits to the right of the comma. Example: The price must not have more than 2 digits after the comma.
Min. digits before comma	Defines the minimum number of digits to the left of the comma. Example: The length must have 4 digits before the comma.
Max. digits before comma	Defines the maximum number of digits to the left of the comma. Example: The length must not have more than 4 digits before the comma.
Min length	Defines the minimum number of characters for a field. Note: In message types with fixed text length, blanks are not counted.
Max length	Defines the maximum number of characters for a field. Note: In message types with fixed text length, blanks are not counted.
Date format	Defines the date format to be used for a field. You may use the following reserved characters (case-sensitive): - YY two-digit year, e.g. 08 - YYYY four-digit year, e.g. 2008 - MM two-digit month, e.g. 09 - DD two-digit day, e.g. 27 - hh two-digit hour, e.g. 23 - mm two-digit minute, e.g. 56 - ss two-digit second, e.g. 37 Beside the reserved date characters, you may use any characters you like, e.g. ., - #. Examples: DD.MM.YYYY YYYY-MM-DD YYYYMMDD You may also specify several date formats and attach them to each other using the pipe character | Example: YYYYMMDDhhmm|YYMMDD
Regular Expression	You may define regular expressions. Please refer to the free jakarta-regexp library: http://jakarta.apache.org/regexp/
Number with leading zeroes	Defines whether the value must be completed with leading zeroes (left-aligned).
Field type	Free-text field describing the field type, currently not checked.
Mandatory	If the filed is mandatory, tick the box.

Name	Field check description
Left-aligned	Defines that the field value must be left-aligned; if a blank is found as the first character on the left, the field will be marked as incorrect.
Right-aligned	Defines that the field value must be right-aligned; if a blank is found as the last character on the right the field will be marked as incorrect.
Forbidden characters	You may specify which characters are not allowed in the field. Enter these characters without any separators and blanks between them (case-sensitive). Example for no umlauts allowed: äÄüÜöÖ
Explicitly allowed characters	Specify explicitly allowed characters; give them without any separators and blanks between them (case-sensitive). Example for an email address: abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_-@.1234567890
Forbidden values	Opposite of 'Explicitly allowed values' - see 'Explicitly allowed values'
Explicitly allowed values	You may define a list of valid values. Examples: A01, A02, B03, X04, X05, Y06, Y07, Y08 and Z09 Alternatively, you can specify a pattern using reserved characters as wild cards: + Only numeric characters @ Only alphabetical characters ? Any character * Any character ensuing or starting Example: Instead of specifying the values A01, A02, B03, etc., you can specify the following pattern: @0+

2.2 Add-ons

2.2.1 Normal Add-on

You can add an add-on (Java code) to each check configuration (e.g. bmw-werk1-vda4913). In an add-on, customer-specific extra checks can be executed. For example, if a T is set in the use tag a 716 record must follow.

The following example is a code serving to compare the target receipt date against the transmission date:

```
/*
 * 11. Check: 712 Eintreffdatum Soll muss >= 711 Übertragungsdatum sein
 */
if  (VDAStatics.LK_712.equalsIgnoreCase(lineKey)&&  VDAStatics.FK_712_EINTREFFDATUM_SOLL.equalsIgnoreCase(fieldKey))
{
    // Format ist in Ordnung, Prüfung gegen Übertragungsdatum:
    String line711 = (String) listOfTextLines.get(0);
    String uebertragungsdatum = StringTools.getSafeSubstr(line711, 33,39, "");
    if(value!=null) {
        if (validator.checkFormat("YYMMDD", value)) {
            try {
                Date valueDate = FORMAT_DATE_YY_MM_DD.parse(value);
                Date uebertragungsdatumDate = FORMAT_DATE_YY_MM_DD.parse(uebertragungsdatum);
                if (valueDate.before(uebertragungsdatumDate)) {
                    listOfErrors = Master.addError(listOfErrors, getAddOnPrefix()
                                    + "EINTREFFDATUM_SOLL_GROESSER",
                                    "Eintreffdatum Soll muss juenger sein als Uebertragungsdatum.");
                }
            } catch (ParseException e) {
                listOfErrors = Master.addError(listOfErrors, "FORM", "format error", "YYMMDD");
            }
        }
    }
}
```

2.2.2 Backend Add-on

The backend add-on uses the same methods as the normal add-on. The special thing about backend add-ons is that by means of them customer-specific backend systems can be accessed.

3 Monitoring

The web-based monitor allows the monitoring of the files (ASNs) checked by MultiParser. Use the search bar at the top to define your search criteria. The more search criteria you use the less results you get.