Record Transformation Language

Joe Justesen

January 28, 2014

The Record Transformation Language (RTL) provides an easy way to validate and transform your data files. Data files consisting of coma separated values (CSV) can be check for data errors, used to validation or transform of other CSV files or transformed into new data files.

Data Types

RTL will support the following data simple types: int, real, string, date, time, datetime and bool; plus the following container types: set, map and array. These containers can hold any simple data type as well as other containers. The data types in a container must be of a uniform type.

Constant dates, times and datetimes are data types distinct from the strings used to represent them and will be represented by strings delimited by the '~' character in ISO 8601 format. For example, a date would be ~2014-06-10~, a time would be ~22:10:00Z~ and a datetime would be a combination of a date and time or full ISO 8601 date ~2012-11-01T06:15:00.00-05:00~

No implicit data type conversion is supported. All data conversions must be performed using the built in functions: to_int, to_real, to_string, to_date, to_time, to_datetime, and to_bool.

Operators

Supported operators are the assignment =; standard mathematical operators -, +, /, *; and comparison operators <, >, <=, >=, ==, !=.

Control Statements

Supported control statements will be if-else; while and for loops. Additionally, a foreach loop will be supported that will iterator over the members of any container.

Variables

Instance variables must be declared, specifying the data type they contain, before being used.

Functions

There are three types of functions in RTL, generic, filter, and transform functions.

A generic function, denoted by the *function* keyword which can contain any number and type of parameters and a single return type. The return data type is specified after the parameter list, if present.

A filter function, denoted by the *filter* keyword, has only a single parameter, a map representing a single record and returns a boolean. Filter functions can only be used in a *check* process. Returning true sends the record to the next stop of the process, false rejects the record, no further processing is performed on that record. Any modification of the record map lost upon return of the function.

A transform function, denoted by the *transform* keyword, has only a single parameter, a map representing a single record, its return value is ignored. Transform functions can only be used in a *trans* process. Transform functions provide the ability to change the contents of a record. All modifications to the record map parameter will be retained.

In all functions, the last expression evaluated is the value returned by the function. The *return* statement causes the function to exit after evaluating the expression after the return. All expressions after a *return* statement in a function must evaluate to the same data type.

Files

Files that will be used as input or output to the processing, can be specified with *file* statement. There are several types of files: input, output, filter, transform and log files. Input and output files are, as their name implies, ordinary CSV files used to read and write data. Filter and transform files are used as inputs to filter and transform processes respectively and must conform to some simple structural rules. Filter files must contain a set of records with a single element. Transform files must contain a set of records with two elements, creating a map between the first elements and the second elements in the file. Log files are written to using the log statement and have a fixed format.

The map records created for each row, all map the header name to the field's constant as a string. There is no attempt to example the data in the field and determine the correct data type. If no header is present in the input file, the map keys will be the number of the column starting with “1”.

Processing Records

A process in RTL is a chain of *check* and *trans* code blocks or functions that are performed in the sequence defined by process list. c*heck* and *trans* statements can be intermingled as needed and the output for this sequence of processing will be written to an output file.

check

In RTL, there are three types of check blocks. First, in line code blocks act as a filter function defined in the *check* statement. The code in the block is executed as if it was as separate function. Second, a filter function can be named. The filter function would be called for each record in the input file. Third, a string and a filter file can be specified; if the map record value for the string is not found in the filter file, the record is rejected.

If any check rejects a record, the sequence of processing for that record is halted and the processing starts again with the next record in the input file.

trans

In RTL, there are three types of trans blocks. First, in line code blocks act as a trans function defined in the *trans* statement. The code in the block is executed as if it was as separate function. Second, a trans function can be named. The trans function would be called for each record in the input file. Third, a string and a trans file can be specified; the map record value for the string is replaced with the corresponding map value generated from the trans file.

A transform process can add or delete columns from the record by using the *add* or *delete* statement.

to

The results of sending the input record through all the check and trans processes is written to the file listed in the *to* statement.

Runtime Environment

RTL will run on the Java Virtual Machine. Source files will compile to Java byte code and be stored in jar files for execution. There will be no support for explicit Java interoperability besides that supported by the language. Data types in the language will be mapped to Java data types were appropriate.

Example

Here is a small example with a single input file, a filter file, a transform file and the resulting output file. The program in example.trl will take the customers.csv file and verify that the DeliveryDate is after Janary 1, 2014 and that the Product is contained in the product.csv file. The Notes field is removed from the input file and the Type values are changed per the data in the days.csv file. The output is written to the results.csv file using '|' to delimit the fields.

customers.csv

```
Customer,DeliveryDate,Product,Type,Amount,UOM,Notes
10889,2014-01-21T14:23:00,LIN,SDAY,5000,GAL,
10289,2014-01-21T16:00:00,LIN,SDAY,1000,GAL,
10453,2014-01-21T17:12:00,BHY,SDAY,2500,M3,
10351,2014-01-21T18:56:00,LIN,MDAY,3000,LBS,Bill by weight
```

product.csv

```
Product
LIN
LOX
LAR
LHE
CO2
```

days.csv

```
From,To
SDAY,Single Day
MDAY,Multi Day
```

example.trl

```
file customers = "./customers.csv"
file product = "./product.csv"
file days = "./days.csv"
file results = "./results.csv" sep=| output
file track = “~/logs/customers.log” format=log

function PassThrough( int value ) int {
      log track “Customer = {0}”, value
      value
}

filter ValidCustomer {
      rec[Customer] > 0
}

transform ExampleTrans {
      rec[Customer] =
          to_string(PassThrough(to_int(rec[Customer]))) # does nothing
}

process customers
    check { to_date(rec[DeliveryDate]) > ~2014-01-01~ } # inline validiation
    check "Product" product                             # lookup validation
    check ValidCustomer                                 # function validation
    trans { delete rec[Notes] }                         # lnline transformation
                                                        # delete Notes column
    trans "Type" days                                   # lookup transformation
    trans ExampleTrans                                  # function transformation
    to results                                          # resulting file
```

results.csv

```
Customer|DeliveryDate|Product|Type|Amount|UOM
10889|2014-01-21T14:23:00|LIN|Single Day|5000|GAL
10289|2014-01-21T16:00:00|LIN|Single Day|1000|GAL
10351|2014-01-21T18:56:00|LIN|Multi Day|3000|LBS
```

customers.log

```
20140-02-01T18:01:33.298 Customer = 10889
20140-02-01T18:01:33.341 Customer = 10289
20140-02-01T18:01:33.793 Customer = 10351
```