cmlcsvc: CSV File and Report Compare Utility Guide and Reference Beta Test

CML00113-00

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
www.codemagus.com

November 20, 2017

Contents

1 Introduction

The Code Magus CSV File and Report Compare Utility compares the contents of two delimited text files and produces an account of the differences. Examples of delimited files include comma separated values (or CSV) files and fixed column tab delimited report files.

Each row of the input files is identified by a row compare key and compared row by row. The input files do not need to be sorted in this order beforehand. One or more named columns can be identified as the row compare key or, alternatively the relative row (or record) number can be used. The cells of either the common non-key columns or a user specified list of columns can be compared.

The Code Magus CSV File and Report Compare Utility runs on Unix and Unix-based platforms, Windows platforms, z/OS USS and Classic MVS environments.

For the rest of this document both input CSV file and input report file will be referred to as just input file. The difference is usually only the column delimiter.

2 Processing

`cmlcsvc` is a command line utility that will compare the contents (cells), under the specification of a row key, of two delimited input files and prints any differences found. For rows found only on one input file it will print the key value, input file row number and which input file it was found on. For rows that have the same key value across the two input files it will print the names and values of those columns that are different between the two input files. The input files are identified as the left or right input file corresponding to left to right order of they are specified as parameters to the utility.

The first row of an input file is taken to be a column header row and the values are used as the column names. If there is no heading row then there is an option to indicate this, in which case the columns are automatically named A→Z,AA→AZ,BA→BZ and on, as is common in spread sheet applications.

The column names, as read in from an input file, should start with a capital letter and be followed by any number of capital letters, numeric characters or underscores. However names that include lower case letters, spaces or hyphens are automatically folded to upper case and spaces and hyphens are converted to an underscore. When referring to columns in the command line parameters, as in sub section 3.2 on page 6, this should be taken into consideration. Other non-conforming characters in a name could cause syntax errors when processing them as an expression.

A number of rows to initially skip may be specified. These rows are skipped for the purposes of the compare and the first row after them is then deemed to be the first row of the file.

One or more columns may be specified as the key of each row and the compare process will be performed on the ordered set of data. The columns do not have to be contiguous or in order; for example the key columns could be B,E,C and A. If no key is specified then the relative row number is used as the key. The first key column to be specified is the most significant portion of the key. Key significance decreases down to the last specified key having the least significance.

A subset of columns to use in the compare may be specified. A key column may not be specified in the list of columns to compare. If no compare columns are specified, then all the non key columns of the left file are used. Consequently, all these columns must also exist in the right hand file.

Both input files should have a physical layout that makes it possible to identify a common key for matching and common columns for comparing when keys match. As long as this holds true the two input files do not have to have exactly the same layout or order of columns.

- All columns used in the key must exist in both input files and have exactly the same names. If the input files do not have a header then they would also occur in the same physical column.

- If the columns to compare are named (in one or more `--column-to-compare` parameter specifications) and a header record identifies the columns by name then they do not need to be in the same physical columnar order in the input files.

The utility will compare the cell contents of like columns and rows with the same key and identify the differences as follows:

1. A keyed row appears in both input files, but cell contents differ.
   The key of the row and the cells with different contents from each of the input files will be reported.
2. A keyed row exists in only one input file.
   The key value and input file line number will be reported along with a message indicating which input file it appears in.
   - An input file may have a non unique key and if there are not the same number of rows with the same key in each of the input files to compare, the extra rows will be seen as only in one input file and reported as per the previous point.
3. A key row appears in both input files and the cells are all the same.
   This will not be reported unless explicitly requested through an option.

3 Synopsis

3.1 Invocation

To obtain immediate help on all the configuration options, invoke the utility with the `--help` option as follows:

```
Code Magus Limited CSV File (or Report) Compare Utility V1.0: build 2017-11-03-16.03.10
[./cmlcsvc] $Id: help.txt,v 1.5 2017/11/07 16:50:18 hayward Exp $
Copyright (c) 2003--2016 by Code Magus Limited. All rights reserved.
[Contact: www.codemagus.com].

Usage: cmlcsvc [OPTIONS]* <left_report_File> <Right_report_File>

  -k, --key-column=<key_column_name>                          Key column name. Repeat for each key
                                                              in decreasing order of significance
  -l, --left-column=<column_name>:<expression>                Add, or overlay, a column in the left
                                                              file using an expression
  -r, --right-column=<column_name>:<expression>               Add, or overlay, a column in the right
                                                              file using an expression
  -c, --column-to-compare=<column_name>                       Restrict the compare to only these
                                                              columns; Repeat for each column
  -t, --column-relative-tolerance=<column_name>:<percent>     cells are equal if within this percent
                                                              of each other
  -a, --column-absolute-tolerance=<column_name>:<amount>      cells are equal if within this
                                                              absolute amount of each other
  -s, --skip-rows={0|<row number>}                            Skip this many rows from both
                                                              reports before starting
  -d, --delimiter={,|[,~|;:\t]}                               Delimiter of columns in input files
  -p, --replace-non-print={|character}                        Replace non printable characters in
                                                              the input files with this value
  -h, --no-column-headings                                    Input report files have no column
                                                              headings
  -e, --report-on-equal-rows                                  Also report on equal rows
  -A, --analysis-csv-file-name={|<filename>}                  Name of output analysis CSV file
  -v, --verbose                                               Verbose processing mode

Help options:
  -?, --help                                                  Show this help message
      --usage                                                 Display brief usage message
```

3.2 Run Time Options

The run time options of `cmlcsvc` are:

- left_report_File
  This is the name of the first CSV or report file to compare[1].
- right_report_File
  This is the name of the second CSV or report file to compare[1].
- `-k, --key-column={|<key_column_name>}`[1]
  The name of one column in the report to use as the key column. If there are more than one key columns, then repeat this parameter to name the subsequent

[1] Column names are folded to uppercase and spaces and hyphens are replaced with underscores

ones. The order on the command line in which they are specified determines the order of significance of each key. For example `-k C -k Z-k AA -k B` means that four columns make up the key with the value of column `C` being the most significant and that of column `B` the least.

- `-l, --left-column=<column_name>:<expression>`[1]
  `-r, --right-column=<column_name>:<expression>`[1]
  Both of these parameters add a new column to or overlay a current column in either the left file or the right file. `<column_name>` names the column and `<expression>` identifies the expression that will be used to derive the value of the each cell in the column. If the name of the column specified corresponds to an existing column on that file then the column is overlaid and the original field name is prefixed with `RPT_`. All column names from the associated input file (left or right), including the `RPT_` prefixed ones, are available as variables to be used in an expression. Refer to the section 4 on page 11 for information on the available built in expressions. This feature is most useful when the format of the column on the left file does not match the format of the column of the right file. For example if on the left file a date is formatted as CCYYMMDD and on the right file as DDMMYY, an expression can be used to overlay either the left or right file's column in order to coerce it to the same format as in the other file and to therefore compare the date using a common format.

- `-c, --column-to-compare=<column_name>`[1]
  The name of one column to compare when a report row key matches from both reports. IF this parameter is not specified then all the non key columns of the left report are used as the list of columns to match. In this case this set of columns must be a subset of the columns in the right hand report. To specify more than one column to compare, repeat this parameter as many times as required.

- `-t, --column-relative-tolerance=<column_name>:<percent>`[1]
  Cells of this column are considered equal if the percentage difference between the left and right numeric value is within the `<percent>` amount. This parameter can be repeated for each column such a rule should apply to. If either the left or right value is not-a-number (NAN) then this rule is not applied and normal match processing of the cells takes place. Only one rule per column is allowed.

  The percentage difference is defined as the absolute difference of the two numbers divided by the average of the two numbers, expressed as a percent. The percent difference is used instead of the percent change, because percent change has a different value when calculating it from left to right as opposed to right to left; Using the difference divided by the average of the two numbers means that there is no need to guess which way to calculate the relative difference and it does not matter which is the larger value.

  The formula, where `L` is the left cell value and `R` is the right cell value, is

$PercentDifference = |\frac{(L-R)}{((L+R)/2)}| \times 100$

- `-a, --column-absolute-tolerance=<column_name>:<amount>`[1]
  Cells of this column are considered equal if the absolute difference between the left and right numeric value is less than the given `<amount>`. This parameter can be repeated for each column such a rule should apply to. If either the left or right value is not-a-number (NAN) then this rule is not applied and normal match processing of the cells takes place. Only one rule per column is allowed.

- `-s, --skip-rows={0|<row number>}`
  Skip this many rows from both reports before locating the header row or the first data row; See `-h, --no-column-headings` in section 3.2 on page 8 for an explanation as to which.

- `-d, --delimiter={,|[,~|;:\t]}`
  This specifies the column delimiter in the input files. The default is a comma as commonly used in comma separated variable (CSV) files. The tab character often allows columnar reports to be generated and it can be specified with `"\t"`. A valid delimiter is one of the following comma, tilde, bar, semi-colon, colon and Tab.

- `-p, --replace-non-print={|character}`
  Replace non printable characters in the input files with this value. This parameter is mainly used to aid reading the difference report produced as these characters can cause printed output to be misaligned.

  Using it may, on the rare occasion, cause the utility to compare keys or columns as equal when they differed by at least one non-printable character.

  For example, the text with characters {`'A',0xA0,'B',0x09`} would not match the text of the same length {`'A',0x09,'B',0x09`} where `0xA0` is a Non breaking space and `0x09` is a tab. On the other hand if an `@` was specified for this parameter both values would become {`'A','@','B','@'`} and would therefore compare equal.

- `-h, --no-column-headings`
  The input files have no column headings. Use this to indicate that there is no header row holding the column names. When set the columns are assigned the names A→Z,AA→ZZ,BA→BZ and on, in the same manner as common spread sheet applications. All rows in the input files are deemed to be data, unless skipped by `-s, --skip-rows` in section 3.2 on page 8

- `-A, --analysis-csv-file-name={|<filename>}`
  Name of the output analysis CSV file. This file holds one row for each column difference found and is delimited with the same delimiters as the input files; in other words with the value in `--delimiter, -d`. The columns in this diagnostic file are:

- Column Name
  The column name of the cell that is different between the left and right input files.
- Key
  The row key of the row.
- Input Report
  This is either Left, Right or left-right if the key matched on both input files.
- Left Value
  The left cell value if applicable.
- Right Value
  The right cell value if applicable.
- Left Row Number
  The input sequence number of the row from the left input file that the different cell was found.
- Right Row Number
  The input sequence number of the row from the right input file that the different cell was found.
- Numeric difference L-R
  If both cells are present and numeric, then this holds the difference when subtracting the right cell from the left.
- Numeric difference R-L
  If both cells are present and numeric, then this holds the difference when subtracting the left cell from the right.
- Left Character Difference
  If both cells are present then this highlights the characters in the left cell that are different to the characters in the same position in the right cell. A '`#`' is placed in all positions where the characters match, unless the contents include a hash in which case the program tries to use in order one of '`@*$=%.`'. If the hash is not used then an appropriate comment is placed in `Left Comment`
- Left Comment
  Holds an appropriate message if the '`#`' character is not used to mask characters in the left cell that are the same in the right cell.
- Right Character Difference
  If both cells are present then this highlights the characters in the right cell that are different to the characters in the same position in the left cell. A '`#`' is placed in all positions where the characters match, unless the contents include a hash in which case the program tries to use in order one of

'`@*$=%.`'. If the hash is not used then an appropriate comment is placed in `Right Comment`

  - Right Comment
    Holds an appropriate message if the '`#`' character is not used to mask characters in the right cell that are the same in the left cell.

- `-e, --report-on-equal-rows`
  Also report on equal rows. If set the utility will also report on all rows that compare equal.
- `-v, --verbose`
  Verbose processing mode. This causes the utility to write program processing information to standard error. This may be useful for debugging issues.
- `-?, --help`
  Show this help message.
- `--usage`
  Display brief usage message.

4 Using Expressions

4.1 Overview

There are many built in expressions that can be used. For more information refer to appendix A.3 on page 39 or see CML00091-01 expeval: Expression Evaluation User Guide[2]

4.2 Examples

A new (virtual) column can be created or a current column can be overlaid by specifying a name and an expression for the column. Each original column from the input files can be used as variables in the expression as well as the following utility defined variables:

- `input_row_number`
  This variable holds the original row on the input file that is currently being processed. The rows from the input file may not be processed in arrival order depending on the key definition for the compare run.
- `generated_key`
  This variable holds the generated key of the input file when no columns are specified as the key. It starts from 1 with the first data row in the input file and is incremented by one on each new row read.

All expressions already defined can be used in the definition of a new column.

For example if an input file has the following columns:

- `KEY` A customer number.
- `SURNAME` The customer's surname.
- `FIRSTNAME` The customer's first name.
- `DOB` The customer's date of birth in the format MMDDYY.
- `AGEGROUP` The customer's age group containing 1, 2, 3 ... 15.

Then the following further columns could be defined:

- `FULLNAME:strcat(FIRSTNAME,strcat(' ',SURNAME))`
  This will concatenate the customers first name, a space and the customer's surname into one column.
- `DATEOFBIRTH:strftime(time2epoch(DOB,'%m%d%Y'),'%Y%m%d')`
  This expression converts the date in DOB to an epoch date (often the number of seconds since 1st January 1970) and then back to an ISO date format that can be used to sort records chronologically.

- `AGES:lookup('AgeGroups.txt',AGEGROUP)"`
  This expression will create a new column using a look up table from the contents of the file `AgeGroups.txt` which must contain one line for each keyword=value. For example the file could contain:

  ```
  0=Under Ten
  1=Teenager
  2=Young Adult
  3=Adult
  4=Adult
  5=Adult
  6=Adult
  7=Pensioner
  8=Pensioner
  9=Pensioner
  10=Centurion
  ```

  and if the value on a row is `7` then the result of the expression and the column value of `AGES` will be `Pensioner`.

- `AGEGROUP:lookup('AgeGroups.txt',RPT_AGEGROUP)"`
  This expression will overlay the column `AGEGROUP` as the name specified is the same as that on the input file. The original field on the input file is automatically renamed to `RPT_AGEGROUP` and can still be used in any expression, including this overlay one. The expression uses the same look up table as the previous example but now for each row the `AGEGROUP` column's content is not a numeric string but the looked up descriptive string.

5 Diagnosing Problems

At times the options given to the program may not match the actual layout of the Report or CSV files. Two examples of these are:

5.1 Not specifying `--no-column-headings`

`-h, --no-column-headings` is not specified when the input files do not contain headers in the first column.

For example, `--key-column=A` is specified and the following error message is received,

```
Error preparing for keyed access. Diagnostic: 1:
                        table sample2_work has no column named A
```

this may be the case and the program has taken the first (data) row as the column names.

5.2 Delimiter is not specified

The input files have a delimiter of `;` and `--delimiter=;` is not specified. This means that the program uses the default delimiter and expects the data to be comma delimited, and consequently reads the whole first row as one field. The error message may be misleading as the file does contain the column you want as the key, but is not recognised due to the column name row being incorrectly parsed.

For example if the following is specified for a test

```
./cmlcsvc -kA-RECORD-TYPE sample.csv sample2.csv
```

and you receive the following error message, which at face value indicates the file does not contain the key `A-RECORD-TYPE`, when in fact it does.

```
Error preparing for keyed access. Diagnostic: 1:
                        table sample2_work has no column named A-RECORD-TYPE
SQL: create  index "sample2_work_index" on "sample2_work" ("A-RECORD-TYPE")
```

To confirm if this is the case, rerun the compare and specify `--verbose`

```
./cmlcsvc --verbose -kA-RECORD-TYPE sample.csv sample2.csv
```

and you should see the following diagnostic indicating that only one column was found and what delimiter was used.

```
Identified 1 Column using delimiter "," in "CSV / Report" sample2.csv
01: A-RECORD-TYPE;A-ALPHA-FIELD-1;A-NUMERIC-FIELD-2;A-NUMERIC-FIELD-3;...
                                ...A-NUMERIC-FIELD-4;A-NUMERIC-FIELD-5;
```

With the right delimiter set and `--verbose` set you would see this output:

```
./cmlcsvc --verbose -d; -kA-RECORD-TYPE sample.csv sample2.csv
...
Identified 7 Columns using delimiter ";" in "CSV / Report" sample2.csv
01: A-RECORD-TYPE
02: A-ALPHA-FIELD-1
03: A-NUMERIC-FIELD-2
04: A-NUMERIC-FIELD-3
05: A-NUMERIC-FIELD-4
06: A-NUMERIC-FIELD-5
07: G
```

6 Bulk processing

A further utility incorporates the **cmlcsvc**: CSV File and Report Compare Utility Guide and Reference Beta Test so that it can compare corresponding files across two folders. Specifically it will compare the latest file in each folder and save the report in a reports folder or email it to one or more recipients as specified.

This means the utility can be configured to run automatically on a server at intervals or only when a new file appears in both folders.

7 Examples

The following sections explain in practical terms how `cmlcsvc` works.

7.1 Example One

7.1.1 Basic Example

This example

- uses the basic two input files as shown in section 8.1 on page 26.
- As no keys are specified, a generic key is created based on the sequence in which data rows are read from the input files. This means that all row keys will match up to the number of rows that are in both input files. Thereafter all the keys left over in one of the input files will be reported as occurring in only that one input file.
- Uses the default option for all the parameters.
    1. No key columns are specified, so a key is generated which is the sequence number of the record as it is read.
    2. No compare columns are named, so the list of column is taken from the columns in the left input file.
    3. The input files start in row 1.
    4. Columns are separated by a comma.
    5. No non printable characters in the input are replaced with a printable substitute.
    6. There is a header row that names the columns.
    7. Equal rows are not reported.

The command:

```
cmlcsvc \
   examples/example_one_left.csv \
   examples/example_one_right.csv \
   >examples/example_one_report.txt
```

produces the result as shown below.

```
cat examples/example_one_report.txt

Keys: generated_sequence:


Item: generated_sequence = 0000000002:
```

```
Report Column Name \                    3 <= Report Row Number => 3
              COL2 |                          r2cx <=> r2c2                          |

Item: generated_sequence = 0000000005:
Report Column Name \                    6 <= Report Row Number => 6
              COL4 |                          r5cx <=> r5c4                          |

Item: generated_sequence = 0000000008:
Report Column Name \                    9 <= Report Row Number => 9
              COL6 |                          r8cx <=> r8c6                          |

Item: Report Row Number 11 only in right file. Key: generated_sequence = 0000000010:

Left file examples/example_one_left.csv
Left file records read   : 9
Rows on left file only   : 0

Right file examples/example_one_right.csv
Right file records read  : 10
Rows on right file only  : 1

Number of rows matched   : 6
Number of rows where key
   matched but not data  : 3
```

7.2 Example Two

7.2.1 Using Keys

This example

- uses the two (keyed) input files as shown in section 8.2 on page 27.
- As a key (two in this case) is defined the order of comparing is by key and not input sequence number. This means that key values could be missing from either input file and not only at the end as with the basic example.
- Uses Two key columns and the default options for the rest of the parameters.
  1. Two key columns are specified.
  2. No compare columns are named, so the list of column is taken from the non key columns in the left input file.
  3. The input files start in row 1.
  4. Columns are separated by a comma.
  5. No non printable characters in the input are replaced with a printable substitute.
  6. There is a header row that names the columns.
  7. Equal rows are not reported.

The command:

```
cmlcsvc \
   --key-column=COL5 \
   --key-column=COL1 \
   examples/example_two_left.csv \
   examples/example_two_right.csv \
   >examples/example_two_A_report.txt
```

produces the result as shown below.

```
cat examples/example_two_A_report.txt

Keys: COL5; COL1:


Item: Report Row Number 2 only in right file. Key: COL5 = r1c5; COL1 = r1c1:

Item: COL5 = r2c5; COL1 = r2c1:
Report Column Name \                    2 <= Report Row Number => 3
              COL2 |                          r2cx <=> r2c2                          |

Item: COL5 = r5c5; COL1 = r5c1:
Report Column Name \                    5 <= Report Row Number => 6
              COL4 |                          r5cx <=> r5c4                          |

Item: Report Row Number 6, only in left file. Key: COL5 = r6c5; COL1 = r6c1:

Item: COL5 = r8c5; COL1 = r8c1:
Report Column Name \                    8 <= Report Row Number => 8
              COL6 |                          r8cx <=> r8c6                          |

Item: Report Row Number 10 only in right file. Key: COL5 = rAc5; COL1 = rAc1:

Left file examples/example_two_left.csv
Left file records read   : 8
Rows on left file only   : 1

Right file examples/example_two_right.csv
Right file records read  : 9
Rows on right file only  : 2

Number of rows matched   : 4
Number of rows where key
   matched but not data  : 3
```

7.2.2 Subset of Columns

This example

- uses the two (keyed) input files as shown in section 8.2 on page 27.
- Uses Two key columns and the default options for the rest of the parameters.
  1. Two key columns are specified.
  2. Compares only the columns `COL2` and `COL6` of the input files.
  3. The input files start in row 1.
  4. Columns are separated by a comma.
  5. No non printable characters in the input are replaced with a printable substitute.
  6. There is a header row that names the columns.
  7. Equal rows are not reported.

The command:

```
cmlcsvc \
   --key-column=COL5 \
   --key-column=COL1 \
   --column-to-compare=COL2 \
   --column-to-compare=COL6 \
```

```
examples/example_two_left.csv \
examples/example_two_right.csv \
>examples/example_two_B_report.txt
```

produces the result as shown below.

```
cat examples/example_two_B_report.txt

Comparing Columns:
COL2; COL6

Keys: COL5; COL1:


Item: Report Row Number 2 only in right file. Key: COL5 = r1c5; COL1 = r1c1:

Item: COL5 = r2c5; COL1 = r2c1:
Report Column Name \                    2 <= Report Row Number => 3
              COL2 |                           r2cx <=> r2c2                           |

Item: Report Row Number 6, only in left file. Key: COL5 = r6c5; COL1 = r6c1:

Item: COL5 = r8c5; COL1 = r8c1:
Report Column Name \                    8 <= Report Row Number => 8
              COL6 |                           r8cx <=> r8c6                           |

Item: Report Row Number 10 only in right file. Key: COL5 = rAc5; COL1 = rAc1:

Left file examples/example_two_left.csv
Left file records read   : 8
Rows on left file only   : 1

Right file examples/example_two_right.csv
Right file records read  : 9
Rows on right file only  : 2

Number of rows matched   : 5
Number of rows where key
   matched but not data  : 2
```

7.3 Example Three

7.3.1 All the options

This example

- uses the two (keyed) input files as shown in section 8.3 on page 27.
- Uses many of the options.
    1. Two key columns are specified. these are `COL5` and `A` which is a generated column name.
    2. Compares a subset of columns including column `AZ` of the input files. Column `AZ` is the 27th column and unnamed in the input files.
    3. The input files start in row 8. The first 7 rows are comments and not part of the report.
    4. Columns are separated by a tab character.
    5. Non printable characters in the input are replaced with a printable substitute.
    6. There is a header row that names the columns.

7. Equal rows are reported.

The command:

```
cmlcsvc \
   --key-column=COL5 \
   --key-column=A \
   --column-to-compare=AA \
   --column-to-compare=COL10 \
   --column-to-compare=COL9 \
   --column-to-compare=COL7 \
   --column-to-compare=COL6 \
   --column-to-compare=COL4 \
   --column-to-compare=COL3 \
   --skip-rows=7 \
   --delimiter="\t" \
   --replace-non-print="@" \
   --report-on-equal-rows \
   examples/example_three_left.csv \
   examples/example_three_right.csv \
   >examples/example_three_A_report.txt
```

produces the result as shown below.

```
cat examples/example_three_A_report.txt

Comparing Columns:
AA; COL3; COL4; COL6; COL7; COL9; COL10

Keys: COL5; A:


Item: COL5 = r1c5; A = r1c1:
Report Column Name \                    9 <= Report Row Number => 9
              COL3 |                           r1c@ <=> r1c3                          |

Item: Report Row Number 10, only in left file. Key: COL5 = r2c5; A = r2c1:

Item: COL5 = r3c5; A = r3c1:
Report Column Name \                   11 <= Report Row Number => 10
              COL4 |                           r3c@ <=> r3c4                          |

Item: Report Row Number 12, only in left file. Key: COL5 = r4c5; A = r4c1:

Item: COL5 = r5c5; A = r5c1:
Report Column Name \                   13 <= Report Row Number => 11
              COL6 |                           r5c@ <=> r5c6                          |
Key: COL5 = r6c5; A = r6c1: Match
               |                  14 <= Report Row Number => 12

Item: COL5 = r7c5; A = r7c1:
Report Column Name \                   15 <= Report Row Number => 13
              COL7 |                           r7c@ <=> r7c7                          |

Item: Report Row Number 14 only in right file. Key: COL5 = r8c5; A = r8c1:

Item: COL5 = r9c5; A = r9c1:
Report Column Name \                   16 <= Report Row Number => 15
              COL9 |                           r9c@ <=> r9c9                          |

Item: COL5 = rac5; A = rac1:
Report Column Name \                   17 <= Report Row Number => 16
             COL10 |                          rac@@ <=> rac10                         |

Left file examples/example_three_left.csv
Left file records read   : 9
Rows on left file only   : 2
Non printable characters replaced:
Character Count
     0xA0     7

Right file examples/example_three_right.csv
Right file records read  : 8
Rows on right file only  : 1

Number of rows matched   : 1
Number of rows where key
   matched but not data  : 6
```

7.3.2 All the options with unnamed columns

This example uses the same input files as the previous one, except that it skips the header row. As there is now no header row in the input files, the option
`-h, --no-column-headings`
is added so that the column names are generated. The final report of the differences should be the same as the one from the previous part of this example except that the column names will be different.

- uses the two (keyed) input files as shown in section 8.3 on page 27.
- Uses many of the options.
    1. Two key columns are specified. these are `E` and `A` which are both generated column names.
    2. Compares a subset of columns. All the column names are generated and must therefore reflect the its physical column position as a letter name.
    3. The input files start in row 9. The first 8 rows are comments and not part of the report. In fact this example ignores row 8, the real column names, in order to simulate an example where no column names are in the input files.
    4. Columns are separated by a tab character.
    5. Non printable characters in the left input file are replaced with a printable substitute.
    6. There is no header row that names the columns.
    7. Equal rows are reported.

The command:

```
cmlcsvc \
   --key-column=E \
   --key-column=A \
   --column-to-compare=AA \
   --column-to-compare=J \
   --column-to-compare=I \
   --column-to-compare=G \
   --column-to-compare=F \
   --column-to-compare=D \
   --column-to-compare=C \
   --skip-rows=8 \
   --delimiter="\t" \
   --replace-non-print="@" \
   --report-on-equal-rows \
   --no-column-headings \
   examples/example_three_left.csv \
   examples/example_three_right.csv \
   >examples/example_three_B_report.txt
```

produces the result as shown below.

```
cat examples/example_three_B_report.txt

Comparing Columns:
C; D; F; G; I; J; AA

Keys: E; A:


Item: E = r1c5; A = r1c1:
Report Column Name \                    9 <= Report Row Number => 9
                 C |                           r1c@ <=> r1c3                           |

Item: Report Row Number 10, only in left file. Key: E = r2c5; A = r2c1:

Item: E = r3c5; A = r3c1:
Report Column Name \                   11 <= Report Row Number => 10
                 D |                           r3c@ <=> r3c4                           |

Item: Report Row Number 12, only in left file. Key: E = r4c5; A = r4c1:

Item: E = r5c5; A = r5c1:
Report Column Name \                   13 <= Report Row Number => 11
                 F |                           r5c@ <=> r5c6                           |
Key: E = r6c5; A = r6c1: Match
                  |                  14 <= Report Row Number => 12

Item: E = r7c5; A = r7c1:
Report Column Name \                   15 <= Report Row Number => 13
                 G |                           r7c@ <=> r7c7                           |

Item: Report Row Number 14 only in right file. Key: E = r8c5; A = r8c1:

Item: E = r9c5; A = r9c1:
Report Column Name \                   16 <= Report Row Number => 15
                 I |                           r9c@ <=> r9c9                           |

Item: E = rac5; A = rac1:
Report Column Name \                   17 <= Report Row Number => 16
                 J |                          rac@@ <=> rac10                          |

Left file examples/example_three_left.csv
Left file records read   : 9
Rows on left file only   : 2
Non printable characters replaced:
Character Count
     0xA0     7

Right file examples/example_three_right.csv
Right file records read  : 8
Rows on right file only  : 1

Number of rows matched   : 1
Number of rows where key
   matched but not data  : 6
```

7.4 Example Four

7.4.1 Comparison Tolerances

This example

- uses the two (keyed) input files as shown in section 8.4 on page 29.
- Uses one key column and the default options for the rest of the parameters.
- Defines two relative tolerances and two absolute tolerances for various columns
- Uses default options for the rest of the parameters.

    1. Shows some rows where cells are within the tolerance specified and so are deemed to be equal.

2. No compare columns are named, so the list of column is taken from the non key columns in the left input file.
3. The input files start in row 1.
4. Columns are separated by a tab.
5. No non printable characters in the input are replaced with a printable substitute.
6. There is a header row that names the columns.
7. Equal rows are not reported.

The command:

```
./cmlcsvc \
   --delimiter=\\t \
   --key-column=COL6 \
   --column-relative-tolerance=COL2:39.5 \
   --column-absolute-tolerance=COL3:6 \
   --column-relative-tolerance=COL4:9 \
   --column-absolute-tolerance=COL5:5 \
   examples/example_four_left.csv \
   examples/example_four_right.csv\
   >examples/example_four_report.txt
```

produces the result as shown below, where

- 3 columns (2,4,5 on rows 3,7,9 respectively) are different because the difference lies outside the tolerance.
- column 7 is different in row 10 without any tolerance.
- column 3 has a difference on row 5, but as it lies within the tolerance specified for it, it is not reported in the difference report.

```
cat examples/example_four_report.txt

Keys: COL6:


Item: COL6 = 10.06:
Report Column Name \                    3 <= Report Row Number => 3
              COL2 |                          10.02 <=> 15.02                          |

Item: COL6 = 50.05:
Report Column Name \                    7 <= Report Row Number => 7
              COL4 |                          50.04 <=> 55.04                          |

Item: COL6 = 70.06:
Report Column Name \                    9 <= Report Row Number => 9
              COL5 |                          70.05 <=> 75.05                          |

Item: COL6 = 90.06:
Report Column Name \                   10 <= Report Row Number => 10
              COL7 |                          90.07 <=> 90.08                          |

Left file examples/example_four_left.csv
Left file records read   : 9
Rows on left file only   : 0

Right file examples/example_four_right.csv
Right file records read  : 9
Rows on right file only  : 0

Number of rows matched   : 5
```

```
Number of rows where key
   matched but not data  : 4
```

7.5 Example Five

7.5.1 Analysis Report

This example

- Uses the same input and options as example four, but writes an analysis report that details the differences in a single CSV file.

The command:

```
./cmlcsvc \
   --delimiter=\\t \
   --key-column=COL6 \
   --column-relative-tolerance=COL2:39.5 \
   --column-absolute-tolerance=COL3:6 \
   --column-relative-tolerance=COL4:9 \
   --column-absolute-tolerance=COL5:5 \
   --analysis-csv-file-name=examples/example_five_analysis.csv \
   examples/example_four_left.csv \
   examples/example_four_right.csv\
   >examples/example_four_report.txt
```

produces the same result as example four, but also the CSV as shown below:

```
cat examples/example_five_analysis.csv

Column Name Key Input Report Left Value Right Value Left Row Number Right Row Number Numeric difference
   L-R Numeric difference R-L Left Character Difference Left Comment Right Character Difference Right Comment
COL2 :10.06: Left and Right 10.02 15.02 3 3 -5.00 5.00 "#0###" "" "#5###" ""
COL4 :50.05: Left and Right 50.04 55.04 7 7 -5.00 5.00 "#0###" "" "#5###" ""
COL5 :70.06: Left and Right 70.05 75.05 9 9 -5.00 5.00 "#0###" "" "#5###" ""
COL7 :90.06: Left and Right 90.07 90.08 10 10 -0.01 0.01 "####7" "" "####8" ""
```

7.6 Example Six

7.6.1 Analysis Report

This example

- uses the two non-keyed input files as shown in section 8.6 on page 30.
- Shows the use of some built in functions to manipulate columns using date and string transformations.
- Uses default options for the rest of the parameters.

Assume that two CSV files are extracts from a database and a transformed copy of that database. The transformed database has standardised dates and times (for example

ISO date and time) and a descriptive string in place of a mnemonic for the column `CATEGORY`.

Looking at only one row of these CSV files, if they are compared directly the differences are very noticeable as below.

```
./cmlcsvc \
   examples/example_six_left.csv\
   examples/example_six_right.csv\
   >examples/example_six_report1.txt
```

which produces the result as shown below.

```
cat examples/example_six_report1.txt

Comparing Columns:
DATE; TIME; CATEGORY

Keys: generated_sequence:


Item: generated_sequence = 0000000001:
Report Column Name \                  2 <= Report Row Number => 2
              DATE |                       031917 <=> 20170319                      |
              TIME |                         03PM <=> 15H00                         |
          CATEGORY |                          lbook <=> LibraryBook                 |

Left file examples/example_six_left.csv
Left file records read   : 1
Rows on left file only   : 0

Right file examples/example_six_right.csv
Right file records read  : 1
Rows on right file only  : 0

Number of rows matched   : 0
Number of rows where key
   matched but not data  : 1
```

Clearly this is not acceptable as these rows are logically the same. By overlaying the left hand file's columns with transformed values, the two rows should now compare equal.

```
./cmlcsvc \
  --left-column="DATE:strcat('20',strcat(substr(RPT_DATE,5,2),\
         strcat(substr(RPT_DATE,1,2),substr(RPT_DATE,3,2))))" \
  --left-column="TIME:strftime(SysTime(strcat('20170101',\
            RPT_TIME),'%Y%d%m%I%p'),'%HH%M')" \
  --left-column="CATEGORY:lookup('lbook=LibraryBook,bbook=BoughtBook',\
            RPT_CATEGORY)" \
   examples/example_six_left.csv\
   examples/example_six_right.csv\
   >examples/example_six_report2.txt
```

The expressions work in the following way:

- The first expression uses `substr` (Appendix A.3.2 on page 39) to extract the `DD, MM and YY` parts of the original value. It then uses `strcat()` (Appendix A.3.5 on page 41) to concatenate them into the correct order. At the same time is assumes the century to be `20`.
- The second expression manipulates a time by first converting the original value to an epoch based time using `SysTime()` (Appendix A.3.12 on page 44). An epoch date is stored as the number of seconds since a known date, usually 1st January 1970. `SysTime` works better if it has a complete date as well as the time. This is why the constant `20170101` is added to the front of the field first. It

then applies the function `strftime()` (Appendix A.3.13 on page 45) to format the epoch date as a time where the hours and minutes are two numbers wide and separated by a capital `H`. Both `SysTime()` and `strftime()` take a second parameter which is a mask that describes the date format. This mask directly relates to the mask used by the C run time programming functions `strptime()` and `strftime()` respectively.

- The third function uses `lookup()` (Appendix A.3.19 on page 51) creates a lookup table using the values of the fist parameter (this is only done on the first call) and then looks up the value of the second parameter and returns the corresponding value. In this example the value `lbook` becomes `LibraryBook`

As can be seen by the output below the two rows now compare as equal.

```
cat examples/example_six_report2.txt

Comparing Columns:
DATE; TIME; CATEGORY

Keys: generated_sequence:


Left file examples/example_six_left.csv
Left file records read   : 1
Rows on left file only   : 0

Right file examples/example_six_right.csv
Right file records read  : 1
Rows on right file only  : 0

Number of rows matched   : 1
Number of rows where key
   matched but not data  : 0
```

8 Example Input Files

8.1 Example One

8.1.1 Left Input File

```
COL1,COL2,COL3,COL4,COL5,COL6,COL7,COL8
r1c1,r1c2,r1c3,r1c4,r1c5,r1c6,r1c7,r1c8
r2c1,r2cx,r2c3,r2c4,r2c5,r2c6,r2c7,r2c8
r3c1,r3c2,r3c3,r3c4,r3c5,r3c6,r3c7,r3c8
r4c1,r4c2,r4c3,r4c4,r4c5,r4c6,r4c7,r4c8
r5c1,r5c2,r5c3,r5cx,r5c5,r5c6,r5c7,r5c8
r6c1,r6c2,r6c3,r6c4,r6c5,r6c6,r6c7,r6c8
r7c1,r7c2,r7c3,r7c4,r7c5,r7c6,r7c7,r7c8
r8c1,r8c2,r8c3,r8c4,r8c5,r8cx,r8c7,r8c8
r9c1,r9c2,r9c3,r9c4,r9c5,r9c6,r9c7,r9c8
```

8.1.2 Right Input File

```
COL1,COL2,COL3,COL4,COL5,COL6,COL7,COL8
r1c1,r1c2,r1c3,r1c4,r1c5,r1c6,r1c7,r1c8
```

```
r2c1,r2c2,r2c3,r2c4,r2c5,r2c6,r2c7,r2c8
r3c1,r3c2,r3c3,r3c4,r3c5,r3c6,r3c7,r3c8
r4c1,r4c2,r4c3,r4c4,r4c5,r4c6,r4c7,r4c8
r5c1,r5c2,r5c3,r5c4,r5c5,r5c6,r5c7,r5c8
r6c1,r6c2,r6c3,r6c4,r6c5,r6c6,r6c7,r6c8
r7c1,r7c2,r7c3,r7c4,r7c5,r7c6,r7c7,r7c8
r8c1,r8c2,r8c3,r8c4,r8c5,r8c6,r8c7,r8c8
r9c1,r9c2,r9c3,r9c4,r9c5,r9c6,r9c7,r9c8
rAc1,rAc2,rAc3,rAc4,rAc5,rAc6,rAc7,rAc8
```

8.2 Example Two

8.2.1 Left Input File

```
COL1,COL2,COL3,COL4,COL5,COL6,COL7,COL8
r2c1,r2cx,r2c3,r2c4,r2c5,r2c6,r2c7,r2c8
r3c1,r3c2,r3c3,r3c4,r3c5,r3c6,r3c7,r3c8
r4c1,r4c2,r4c3,r4c4,r4c5,r4c6,r4c7,r4c8
r5c1,r5c2,r5c3,r5cx,r5c5,r5c6,r5c7,r5c8
r6c1,r6c2,r6c3,r6c4,r6c5,r6c6,r6c7,r6c8
r7c1,r7c2,r7c3,r7c4,r7c5,r7c6,r7c7,r7c8
r8c1,r8c2,r8c3,r8c4,r8c5,r8cx,r8c7,r8c8
r9c1,r9c2,r9c3,r9c4,r9c5,r9c6,r9c7,r9c8
```

8.2.2 Right Input File

```
COL1,COL2,COL3,COL4,COL5,COL6,COL7,COL8
r1c1,r1c2,r1c3,r1c4,r1c5,r1c6,r1c7,r1c8
r2c1,r2c2,r2c3,r2c4,r2c5,r2c6,r2c7,r2c8
r3c1,r3c2,r3c3,r3c4,r3c5,r3c6,r3c7,r3c8
r4c1,r4c2,r4c3,r4c4,r4c5,r4c6,r4c7,r4c8
r5c1,r5c2,r5c3,r5c4,r5c5,r5c6,r5c7,r5c8
r7c1,r7c2,r7c3,r7c4,r7c5,r7c6,r7c7,r7c8
r8c1,r8c2,r8c3,r8c4,r8c5,r8c6,r8c7,r8c8
r9c1,r9c2,r9c3,r9c4,r9c5,r9c6,r9c7,r9c8
rAc1,rAc2,rAc3,rAc4,rAc5,rAc6,rAc7,rAc8
```

8.3 Example Three

Rows in the example input files may be shown wrapped. In the actual input file there is no new line at that point as this is just the display representation of the input file.

8.3.1 Left Input File

It is not possible to display the `0xA0` characters in the input file. This is the same input file but with the `0xA0` characters rendered as question marks. When looking at the difference report these are shown as '@', because the non printable characters have been substituted.

```
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
COL2 COL3 COL4 COL5 COL6 COL7 COL8 COL9 COL10 COL11 COL12 COL13 COL14 COL15 COL16 COL17 COL18
COL19 COL20 COL21 COL22 COL23 COL24 COL25 COL26
r1c1 r1c2 r1c? r1c4 r1c5 r1c6 r1c7 r1c8 r1c9 r1c10 r1c11 r1c12 r1c13 r1c14 r1c15 r1c16 r1c17
r1c18 r1c19 r1c20 r1c21 r1c22 r1c23 r1c24 r1c25 r1c26 r1c27
r2c1 r2c2 r2c3 r2c4 r2c5 r2c6 r2c7 r2c8 r2c9 r2c10 r2c11 r2c12 r2c13 r2c14 r2c15 r2c16 r2c17
r2c18 r2c19 r2c20 r2c21 r2c22 r2c23 r2c24 r2c25 r2c26 r2c27
r3c1 r3c2 r3c3 r3c? r3c5 r3c6 r3c7 r3c8 r3c9 r3c10 r3c11 r3c12 r3c13 r3c14 r3c15 r3c16 r3c17
r3c18 r3c19 r3c20 r3c21 r3c22 r3c23 r3c24 r3c25 r3c26 r3c27
r4c1 r4c2 r4c3 r4c4 r4c5 r4c6 r4c7 r4c8 r4c9 r4c10 r4c11 r4c12 r4c13 r4c14 r4c15 r4c16 r4c17
r4c18 r4c19 r4c20 r4c21 r4c22 r4c23 r4c24 r4c25 r4c26 r4c27
r5c1 r5c2 r5c3 r5c4 r5c5 r5c? r5c7 r5c8 r5c9 r5c10 r5c11 r5c12 r5c13 r5c14 r5c15 r5c16 r5c17
r5c18 r5c19 r5c20 r5c21 r5c22 r5c23 r5c24 r5c25 r5c26 r5c27
r6c1 r6c2 r6c3 r6c4 r6c5 r6c6 r6c7 r6c8 r6c9 r6c10 r6c11 r6c12 r6c13 r6c14 r6c15 r6c16 r6c17
r6c18 r6c19 r6c20 r6c21 r6c22 r6c23 r6c24 r6c25 r6c26 r6c27
r7c1 r7c2 r7c3 r7c4 r7c5 r7c6 r7c? r7c8 r7c9 r7c10 r7c11 r7c12 r7c13 r7c14 r7c15 r7c16 r7c17
r7c18 r7c19 r7c20 r7c21 r7c22 r7c23 r7c24 r7c25 r7c26 r7c27
r9c1 r9c2 r9c3 r9c4 r9c5 r9c6 r9c7 r9c8 r9c? r9c10 r9c11 r9c12 r9c13 r9c14 r9c15 r9c16 r9c17
r9c18 r9c19 r9c20 r9c21 r9c22 r9c23 r9c24 r9c25 r9c26 r9c27
rac1 rac2 rac3 rac4 rac5 rac6 rac7 rac8 rac9 rac?? rac11 rac12 rac13 rac14 rac15 rac16 rac17
rac18 rac19 rac20 rac21 rac22 rac23 rac24 rac25 rac26 rac27
```

8.3.2 Right File

```
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
This is a test report file where the first 8 rows could be report headings
COL2 COL3 COL4 COL5 COL6 COL7 COL8 COL9 COL10 COL11 COL12 COL13 COL14 COL15 COL16 COL17 COL18
COL19 COL20 COL21 COL22 COL23 COL24 COL25 COL26
r1c1 r1c2 r1c3 r1c4 r1c5 r1c6 r1c7 r1c8 r1c9 r1c10 r1c11 r1c12 r1c13 r1c14 r1c15 r1c16 r1c17
r1c18 r1c19 r1c20 r1c21 r1c22 r1c23 r1c24 r1c25 r1c26 r1c27
r3c1 r3c2 r3c3 r3c4 r3c5 r3c6 r3c7 r3c8 r3c9 r3c10 r3c11 r3c12 r3c13 r3c14 r3c15 r3c16 r3c17
r3c18 r3c19 r3c20 r3c21 r3c22 r3c23 r3c24 r3c25 r3c26 r3c27
r5c1 r5c2 r5c3 r5c4 r5c5 r5c6 r5c7 r5c8 r5c9 r5c10 r5c11 r5c12 r5c13 r5c14 r5c15 r5c16 r5c17
r5c18 r5c19 r5c20 r5c21 r5c22 r5c23 r5c24 r5c25 r5c26 r5c27
r6c1 r6c2 r6c3 r6c4 r6c5 r6c6 r6c7 r6c8 r6c9 r6c10 r6c11 r6c12 r6c13 r6c14 r6c15 r6c16 r6c17
r6c18 r6c19 r6c20 r6c21 r6c22 r6c23 r6c24 r6c25 r6c26 r6c27
r7c1 r7c2 r7c3 r7c4 r7c5 r7c6 r7c7 r7c8 r7c9 r7c10 r7c11 r7c12 r7c13 r7c14 r7c15 r7c16 r7c17
r7c18 r7c19 r7c20 r7c21 r7c22 r7c23 r7c24 r7c25 r7c26 r7c27
r8c1 r8c2 r8c3 r8c4 r8c5 r8c6 r8c7 r8c8 r8c9 r8c10 r8c11 r8c12 r8c13 r8c14 r8c15 r8c16 r8c17
r8c18 r8c19 r8c20 r8c21 r8c22 r8c23 r8c24 r8c25 r8c26 r8c27
r9c1 r9c2 r9c3 r9c4 r9c5 r9c6 r9c7 r9c8 r9c9 r9c10 r9c11 r9c12 r9c13 r9c14 r9c15 r9c16 r9c17
r9c18 r9c19 r9c20 r9c21 r9c22 r9c23 r9c24 r9c25 r9c26 r9c27
rac1 rac2 rac3 rac4 rac5 rac6 rac7 rac8 rac9 rac10 rac11 rac12 rac13 rac14 rac15 rac16 rac17
rac18 rac19 rac20 rac21 rac22 rac23 rac24 rac25 rac26 rac27
```

8.4 Example Four

Rows in the example input files may be shown wrapped. In the actual input file there is no new line at that point as this is just the display representation of the input file.

8.4.1 Left Input File

```
COL2 COL3 COL4 COL5 COL6 COL7 COL8 COL9 COL10 COL11 COL12 COL13 COL14 COL15 COL16 COL17 COL18
COL19 COL20 COL21 COL22 COL23 COL24 COL25 COL26
00.01 00.02 00.03 00.04 00.05 00.06 00.07 00.08 00.09 00.10 00.11 00.12 00.13 00.14 00.15
00.16 00.17 00.18 00.19 00.20 00.21 00.22 00.23 00.24 00.25 00.26 00.27
10.01 10.02 10.03 10.04 10.05 10.06 10.07 10.08 10.09 10.10 10.11 10.12 10.13 10.14 10.15
10.16 10.17 10.18 10.19 10.20 10.21 10.22 10.23 10.24 10.25 10.26 10.27
20.01 20.02 20.03 20.04 20.05 20.06 20.07 20.08 20.09 20.10 20.11 20.12 20.13 20.14 20.15
20.16 20.17 20.18 20.19 20.20 20.21 20.22 20.23 20.24 20.25 20.26 20.27
30.01 30.02 30.03 30.04 30.05 30.06 30.07 30.08 30.09 30.10 30.11 30.12 30.13 30.14 30.15
30.16 30.17 30.18 30.19 30.20 30.21 30.22 30.23 30.24 30.25 30.26 30.27
40.01 40.02 40.03 40.04 40.05 40.06 40.07 40.08 40.09 40.10 40.11 40.12 40.13 40.14 40.15
40.16 40.17 40.18 40.19 40.20 40.21 40.22 40.23 40.24 40.25 40.26 40.27
50.01 50.02 50.03 50.04 50.05 50.05 50.07 50.08 50.09 50.10 50.11 50.12 50.13 50.14 50.15
50.16 50.17 50.18 50.19 50.20 50.21 50.22 50.23 50.24 50.25 50.26 50.27
60.01 60.02 60.03 60.04 60.05 60.06 60.07 60.08 60.09 60.10 60.11 60.12 60.13 60.14 60.15
60.16 60.17 60.18 60.19 60.20 60.21 60.22 60.23 60.24 60.25 60.26 60.27
70.01 70.02 70.03 70.04 70.05 70.06 70.07 70.08 70.09 70.10 70.11 70.12 70.13 70.14 70.15
70.16 70.17 70.18 70.19 70.20 70.21 70.22 70.23 70.24 70.25 70.26 70.27
90.01 90.02 90.03 90.04 90.05 90.06 90.07 90.08 90.09 90.10 90.11 90.12 90.13 90.14 90.15
90.16 90.17 90.18 90.19 90.20 90.21 90.22 90.23 90.24 90.25 90.26 90.27
```

8.4.2 Right Input File

```
COL2 COL3 COL4 COL5 COL6 COL7 COL8 COL9 COL10 COL11 COL12 COL13 COL14 COL15 COL16 COL17 COL18
COL19 COL20 COL21 COL22 COL23 COL24 COL25 COL26
00.01 00.02 00.03 00.04 00.05 00.06 00.07 00.08 00.09 00.10 00.11 00.12 00.13 00.14 00.15
00.16 00.17 00.18 00.19 00.20 00.21 00.22 00.23 00.24 00.25 00.26 00.27
10.01 15.02 10.03 10.04 10.05 10.06 10.07 10.08 10.09 10.10 10.11 10.12 10.13 10.14 10.15
10.16 10.17 10.18 10.19 10.20 10.21 10.22 10.23 10.24 10.25 10.26 10.27
20.01 20.02 20.03 20.04 20.05 20.06 20.07 20.08 20.09 20.10 20.11 20.12 20.13 20.14 20.15
20.16 20.17 20.18 20.19 20.20 20.21 20.22 20.23 20.24 20.25 20.26 20.27
30.01 30.02 35.03 30.04 30.05 30.06 30.07 30.08 30.09 30.10 30.11 30.12 30.13 30.14 30.15
30.16 30.17 30.18 30.19 30.20 30.21 30.22 30.23 30.24 30.25 30.26 30.27
40.01 40.02 40.03 40.04 40.05 40.06 40.07 40.08 40.09 40.10 40.11 40.12 40.13 40.14 40.15
40.16 40.17 40.18 40.19 40.20 40.21 40.22 40.23 40.24 40.25 40.26 40.27
50.01 50.02 50.03 55.04 50.05 50.05 50.07 50.08 50.09 50.10 50.11 50.12 50.13 50.14 50.15
50.16 50.17 50.18 50.19 50.20 50.21 50.22 50.23 50.24 50.25 50.26 50.27
60.01 60.02 60.03 60.04 60.05 60.06 60.07 60.08 60.09 60.10 60.11 60.12 60.13 60.14 60.15
60.16 60.17 60.18 60.19 60.20 60.21 60.22 60.23 60.24 60.25 60.26 60.27
70.01 70.02 70.03 70.04 75.05 70.06 70.07 70.08 70.09 70.10 70.11 70.12 70.13 70.14 70.15
70.16 70.17 70.18 70.19 70.20 70.21 70.22 70.23 70.24 70.25 70.26 70.27
90.01 90.02 90.03 90.04 90.05 90.06 90.08 90.08 90.09 90.10 90.11 90.12 90.13 90.14 90.15
90.16 90.17 90.18 90.19 90.20 90.21 90.22 90.23 90.24 90.25 90.26 90.27
```

8.5 Example Five

This example uses the same input files as those used in example four.

8.6 Example Six

This is a one row example in order to easily show date and string manipulation.

8.6.1 Left Input File

```
DATE,TIME,CATEGORY
031917,03PM,lbook
```

8.6.2 Right Input File

```
DATE,TIME,CATEGORY
20170319,15H00,LibraryBook
```

A Built in Functions

A.1 Expression Overview

The lexical elements of an expression are the variables, literals, operators and other character symbols used to form an expression. These lexical elements or tokens are separated by white spaces. White spaces include sequences of the space character, newline character, the tab character and the linefeed character and their only function is to separate or delimit the tokens.

The lexical elements are often single characters having their own apparent meaning, but some are grouped together to form a word having a specific meaning. Included or associated with each token may be an attribute value.

An expression, made up of the constituent tokens into the syntax and semantics of the grammar, is then validated and evaluated by the expression evaluation library. The evaluation of an expression produces a value that can then be used within the context of the grammar of the specific Code Magus product within which it is specified.

Examples of expressions are:

1. `3+4`
2. `balance + 100`
3. `(account.balance >= 2000)`
4. `where (account.balance = 0)`
5. ```
   where (account.balance < 0 ) and
         (account.overdraft_facility = 'Y')
   ```
6. `SysString(account.balance)`

A.2 Expression Grammar

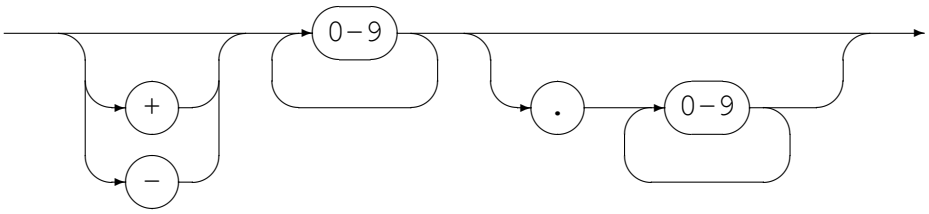
A.2.1 Lexical Elements

The base elements are *Literals* and *Identifiers*.

- Numeric Literals

  A Numeric literal is made up from an optional plus or minus sign followed by one or more digits and optionally followed by a point and one or more digits.
   ```

Number Literal



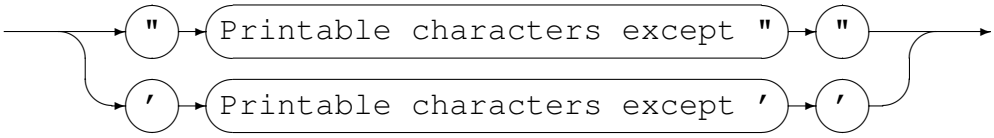
- String Literals

  String literals are made up from

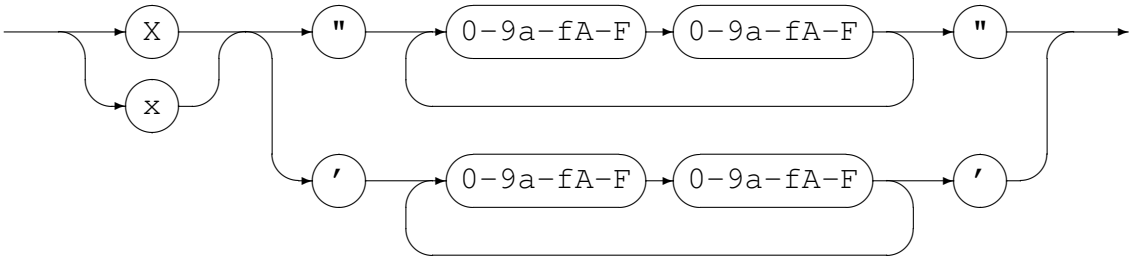
  – Any number of printable characters, except the enclosing character and a newline, enclosed in either single or double quotes.

  – An even number of hexadecimal digits enclosed in either single or double quotes and prefixed with a lower or upper case `X`.

  String Literal



  Hexadecimal Literal



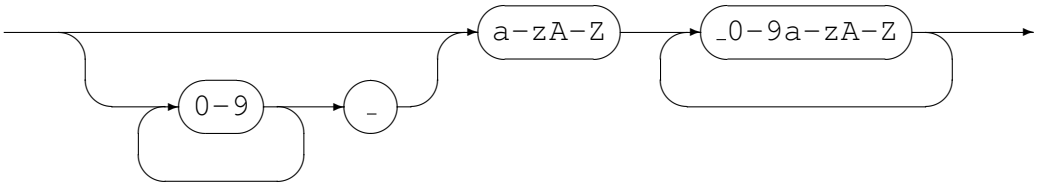
- Identifiers

  An identifier is used for both variable and function names. An identifier must conform to:

  – A lower or upper case alphabetic character followed by any number of underscores, decimal digits and upper and lower case alphabetic characters.

  – One or more decimal digits followed by an underscore and the above rule.

Identifier

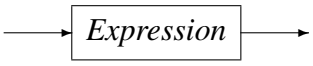


A.2.2 Syntactical Elements

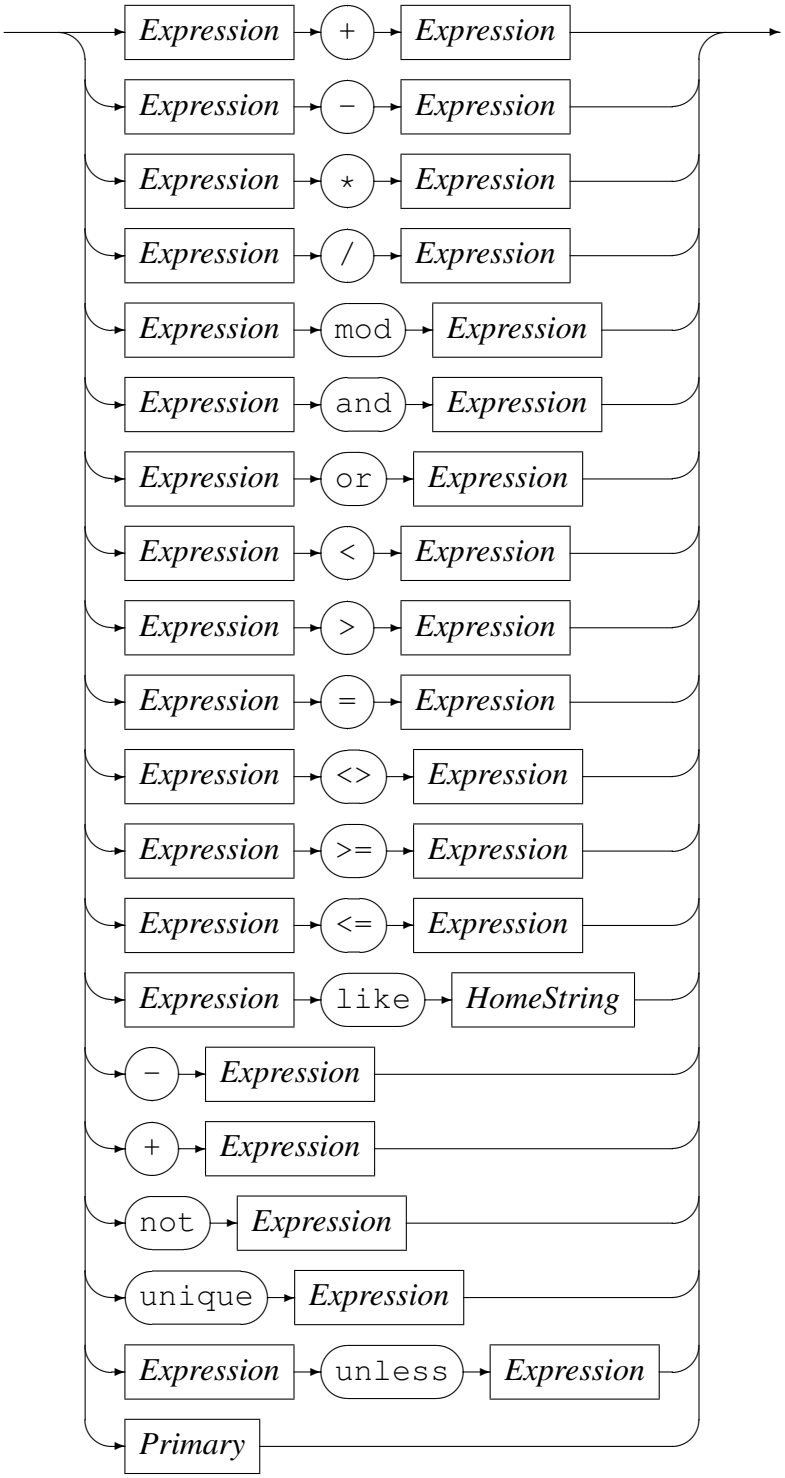
Expressions may themselves be used as syntactical elements when forming a compound expression.

The complete syntax of a compound expression is explained in the following sections starting with the compound expression and working down to the lowest level syntactic element.

CompoundExpression



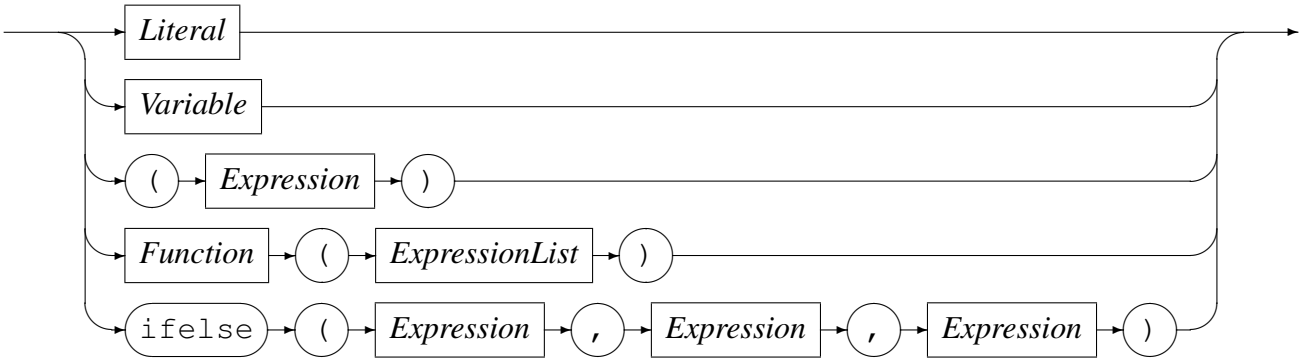
Expression



The `unless`-operator conditionally returns the value of the right-hand operand, unless there is an error evaluating the right-hand operand. In the case where the right-hand operand fails to evaluate to a proper value, the value of the left-hand operand is returned

instead. The left-hand operand is always evaluated before the right-hand operand. If the left-hand operand fails to evaluate to a proper value, then the result of the `unless`-operator is a failure.

Primary



As a terminal in the syntax structure an expression or *Primary* is either a *Literal* or a *Variable*, an *Expression* enclosed in parenthesis, a *Function* call reference, or the conditional evaluation operator `ifelse`. A *Literal* may be a *String Literal* or a *Number Literal* as described in Section A.2.1 on page 32.

Where required by the encoding indicated or defaulted, characters representing the attribute value of a string are changed to an alternate character set if the required character set is not the same as the home character set being used. For example, on a machine in which the characters are naturally represented using the EBCDIC character set encoding (such as code page of 1047 or Latin 1/Open Systems), if the data being processed is from a machine in which the characters are naturally represented using the ASCII character set (such as ISO8859-1), then the characters in the String literal (assumed to be represented in EBCDIC) will be translated to their corresponding ASCII characters for processing. This does not apply to String literals that were represented as a sequence of hexadecimal digits.

Both a *Function* (see Section A.2.2 on page 38) and an *Expression* are made up of sub-expressions, although eventually even they must terminate and resolve to a value.

A *HomeString* is a *String Literal* that may not be represented as a sequence of hexadecimal digits, but in which the encoding is left in the natural encoding of the machine processing the data; that is the machine on which the expression string is being compiled. This is required for the right-hand operand of the like operator as this operator translates the value of the left-hand operand into the local encoding when performing pattern matching.

Operators, variables and functions are described in more detail below:

- Operators

  In the context of the expression evaluation library, an operator is a symbol that

operates on or causes an action to be be performed on the constants and variables adjacent to it. An operator is either

- Monadic
  A monadic operator only operates on one value and usually employ either prefix or postfix notation in that they either occur before or after the value they operate on. The expression evaluation library uses only prefix monadic operators.
- Dyadic
  Dyadic operators operate on two values and employ infix notation in that they operate on the the values that immediately precede and follow the operator.

All operators return a value of a defined type which is the result of the computation. The type returned by an operator must be semantically consistent within the context of the rest of the expression and the grammar it may be embedded in.

Table 1 on page 37 lists the allowed operators, their precedence, associativity, arity (whether or not they are monadic or dyadic) and Type.

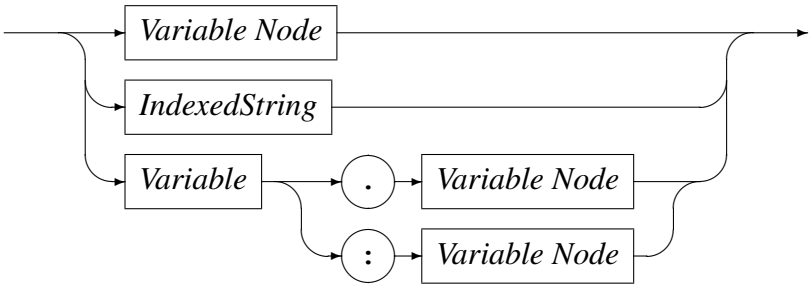
Operator	Precedence	Associativity	Arity	Type
`like`	1	non-assoc	dyadic	Relational
`<>`	1	left	dyadic	Relational
`>=`	1	left	dyadic	Relational
`<=`	1	left	dyadic	Relational
`=`	1	left	dyadic	Relational
`>`	1	left	dyadic	Relational
`<`	1	left	dyadic	Relational
`+`	2	left	dyadic	Arithmetic
`-`	2	left	dyadic	Arithmetic
`or`	2	left	dyadic	Boolean
`*`	3	left	dyadic	Arithmetic
`/`	3	left	dyadic	Arithmetic
`div`	3	left	dyadic	Arithmetic
`and`	3	left	dyadic	Boolean
`mod`	3	left	dyadic	Arithmetic
`-`	4	left	monadic	Arithmetic
`not`	4	left	monadic	Boolean
`unique`	4	left	monadic	boolean
`unless`	5	left	dyadic	boolean

Table 1: Operators: Precedence, Associativity, Arity and Type

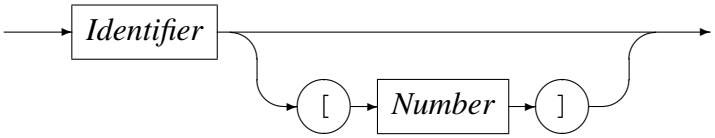
- Variables

A variable is the name of a storage location that holds a value. Simply this name is just an *Identifier*, but may be more than one level or node including an index.

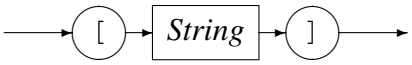
Variable



Variable Node



IndexedString



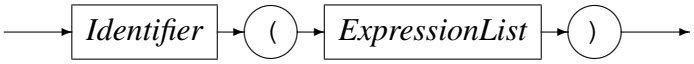
Examples of variable names are:

- `Address` - A single node variable with no indexing.
- `Customer.Address` - A two node variable.
- `Customer.Address[1]` - A two node variable where the Address portion of the variable is the first of an array of items. Here this may be the first line of an address.
- `Customer[3].Address[1]` - A two node variable that specifies the third entry of the Customer array and the first entry of the Address array within that Customer.
- `Customer.Contact.HomePhone` - A three node variable.

- Functions

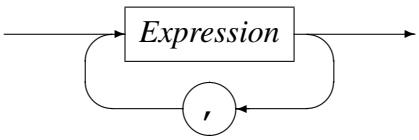
  A function is a special type of operator. It is specified by the function name, an identifier, followed by a comma separated list of arguments enclosed in parentheses.

  Function



where an expression list is defined as

ExpressionList



The function call is replaced with the result of the call and the result type must be semantically consistent within the context of the rest of the expression and the grammar it may be embedded in.

A.3 Built-in Functions

Functions for expression evaluation can be supplied by the application that uses it and as such has a rich set of plug in functions that can not be documented here. However there are functions that are common to all data processing and these are supplied by the expression evaluation library and are described below.

A.3.1 SysStrLen, strlen, length

- **Synopsis**
  - `SysStrLen(string)`
  - `strlen(string)`
  - `length(string)`
- **Parameters**
  - Parameter 1 type: String.
- **Description**

  The `SysStrLne` function (aliasses `strlen`, `length`) returns the number of characters in the string supplied as the first argument.

A.3.2 SysSubStr, substr

- **Synopsis**
  - `SysSubStr(string,start,length)`
  - `substr(string,start,length)`
- **Parameters**

  - Parameter 1 type: String.
  - Parameter 2 type: Number.
  - Parameter 3 type: Number.
  - Return type: String.

- **Description**

  The `SysSubStr` function (alias `substr`) returns a substring of the given string from start for length characters or the remainder of string whichever is the shortest.

  The start must be greater than zero and the length must be zero or greater. If the start position is past the end of the string then a NULL string is returned.

A.3.3 SysString, string

- **Synopsis**
  - `SysString(number)`
  - `string(number)`
- **Parameters**
  - Parameter 1 type: Number.
  - Return type: String.
- **Description** The `SysString` function (alias `string`) returns the value of number as a string.

A.3.4 SysNumber, number

- **Synopsis**
  - `SysNumber(string)`
  - `number(string)`
- **Parameters**
  - Parameter 1 type: String.
  - Return type: Number.
- **Description** The `SysNumber` function (alias `number`) returns a number equivalent to the value of string.

A.3.5 SysStrCat, strcat

- **Synopsis**
  - `SysStrCat(first,second)`
  - `strcat(first,second)`
- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: String.
  - Return type: String.
- **Description** The `SysStrCat` function (alias `strcat`) returns a String which is the concatenation of the two input strings `first` and `second`.

A.3.6 SysStrStr, strstr

- **Synopsis**
  - `SysStrStr(haystack,needle)`
  - `strstr(haystack,needle)`
- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: String.
  - Return type: Number.
- **Description** The `SysStrStr` function (alias `strstr`) returns the start position of `needle` within `haystack`. If `needle` does not occur in `haystack` then zero is returned, otherwise the position (origin 1) is returned.

A.3.7 SysStrSpn, strspn

- **Synopsis**
  - `SysStrSpn(string,accept)`
  - `strspn(string,accept)`
- **Parameters**
  - Parameter 1 type: String.

  - Parameter 2 type: String.
  - Return type: Number.

- **Description** The `SysStrSpn` function (alias `strspn`) returns the number of characters (bytes) in the initial segment of `string` which consist only of characters from `accept`.

A.3.8 SysStrCspn, strcspn

- **Synopsis**
  - `SysStrCspn(string,reject)`
  - `strcspn(string,reject)`
- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: String.
  - Return type: Number.
- **Description** The `SysStrCspn` function (alias `strcspn`) returns the number of characters (bytes) in the initial segment of `string` which do not match any character from `reject`.

A.3.9 SysStrPadRight, padright

- **Synopsis**
  - `SysStrPadRight(string,length,pad)`
  - `padright(string,length,pad)`
- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: Number.
  - Parameter 3 type: String. Although the type is String, only the first character is used as the pad character.
  - Return type: String.
- **Description** The `SysStrPadRight` function (alias `padright`) returns a string whose length is `length` and:

  - if `length` is greater than the length of `string`, is `string` padded on the right with the `pad` character
  - if `length` is less than the length of `string`, is `string` truncated from the right to `length`.
  - if `length` is equal to the length of `string`, is `string`.

A.3.10 SysStrPadLeft, padleft

- **Synopsis**
  - `SysStrPadLeft(string,length,pad)`
  - `padleft(string,length,pad)`
- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: Number.
  - Parameter 3 type: String. Although the type is String, only the first character is used as the pad character.
  - Return type: String.
- **Description** The `SysStrPadLeft` function (alias `padleft`) returns a string whose length is `length` and:
  - if `length` is greater than the length of `string`, is `string` padded on the left with the `pad` character
  - if `length` is less than the length of `string`, is `string` truncated from the left to `length`.
  - if `length` is equal to the length of `string`, is `string`.

A.3.11 SysFmtCurrTime, strftimecurr

- **Synopsis**
  - `SysFmtCurrTime(format)`
  - `strftimecurr(format)`
- **Parameters**
  - Parameter 1 type: String.
  - Return type: String.

- **Description** The `SysFmtCurrTime` function (alias `strftimecurr`) returns a string that represents the current time as formatted according to `format` using the C run-time `strftime()` function. Common values for `format` are:
  - %c - The preferred date and time representation for the current locale.
  - %d - The day of the month as a decimal number (range 01 to 31).
  - %F - Equivalent to %Y-%m-%d (the ISO 8601 date format).
  - %H - The hour as a decimal number using a 24-hour clock (range 00 to 23).
  - %j - The day of the year as a decimal number (range 001 to 366).
  - %m - The month as a decimal number (range 01 to 12).
  - %M - The minute as a decimal number (range 00 to 59).
  - %s - The number of seconds since the Epoch, 1970-01-01 00:00:00
  - %S - The second as a decimal number (range 00 to 60, allows for leap seconds).
  - %T - The time in 24-hour notation (%H:%M:%S).
  - %y - The year as a decimal number without a century (range 00 to 99).
  - %Y - The year as a decimal number including the century.
  - %% - A literal '%' character.
  - Any other characters, not specified by `strftime()`, are copied verbatim from `format` to the result string.

A.3.12 SysTime, time2epoch

- **Synopsis**
  - `SysTime(datetime,format)`
  - `time2epoch(datetime,format)`
- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: String. Default "%Y%m%d".
  - Return type: Number.
- **Description** The `SysTime` function (alias `time2epoch`) returns the number seconds since the Epoch calculated from `datetime` under the specification of `format`.

The seconds since the Epoch, when interpreted as an absolute time value, represents the number of seconds elapsed since the Epoch, `1970-01-01 00:00:00 +0000 (UTC)`.

`datetime` must be a string representation of a date and / or time and `format` must be a date format string that exactly describes `datetime` using the format characters as specified and used by the C function `strptime()`.

Common options for the `format` are:

- %% - The % character.
- %c - The date and time representation for the current locale.
- %C - The century number (0-99).
- %d or %e - The day of month (1-31).
- %H - The hour (0-23).
- %I - The hour on a 12-hour clock (1-12).
- %j - The day number in the year (1-366).
- %m - The month number (1-12).
- %M - The minute (0-59).
- %p - The locale's equivalent of AM or PM. (Note: there may be none.)
- %S - The second (0-60; 60 may occur for leap seconds; earlier also 61 was allowed).
- %T - Equivalent to %H:%M:%S.
- %x - The date, using the locale's date format.
- %X - The time, using the locale's time format.
- %y - The year within century (0-99). When a century is not otherwise specified, values in the range 69-99 refer to years in the twentieth century (1969-1999); values in the range 00-68 refer to years in the twenty-first century (2000-2068).
- %Y - The year, including century (for example, 1991).

A.3.13 SysStrFTime, strftime

- **Synopsis**
  - `SysStrFTime(seconds,format)`
  - `strftime(seconds,format)`

- **Parameters**
  - Parameter 1 type: Number.
  - Parameter 2 type: String.
  - Return type: String.
- **Description** The `SysStrFTime` function (alias `strftime`) returns a string date time representation of `seconds` formatted according to `format` as described in the C runtime function `strftime()`.

  `seconds` is the number of seconds since the Epoch, which when interpreted as an absolute time value, represents the number of seconds elapsed since the Epoch, `1970-01-01 00:00:00 +0000 (UTC)`.

  `format` must be a date format string used to format the returned date time string. For common values of `format` see section A.3.11 on page 44

A.3.14 SysInTable, intable

- **Synopsis**
  - `SysInTable(table,search)`
  - `intable(table,search)`
- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: String.
  - Return type: Boolean.
- **Description**

  The `SysInTable` function (alias `intable`) returns a boolean `TRUE` if the value of `search` is found in the table of items `table`, otherwise it returns a boolean `FALSE`.

  The value of `table` may be either the name of a text file in which each line is one element of the table, or a comma (,) or semi-colon (;) delimited string of the element values of the table.
- **Examples**
  - SysInTable("C:\customerNames.txt","Smith") This will test wether the name "Smith" occurs in the list of elements in the file `C:\customerNames.txt`.

- SysInTable("/tmp/customerNames.txt",Record.Surname) This will test whether the name identified by the object types[1] field `Record.Surname` occurs in the list of elements in the file `/tmp/customerNames.txt`.
- SysInTable("Smith,Jones,Right",Record.Surname) This will test whether the name identified by the object types[1] field `Record.Surname` occurs in the list of elements in the comma separated list specified by the first argument.

A.3.15 SysStrCondPack, condpack

- **Synopsis**
  - `SysStrCondPack(String,String)`
  - `condpack(String,String)`
- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: String.
  - Return type: String.
- **Description**

  The `SysStrCondPack` function (alias `condpack`) returns a string which is conditionally formed by packing the string passed in the first parameter using the second parameter as a possible replacement character. If the first parameter matches the regular expression `X"[0-9][A-F][a-f]"` then the hexadecimal characters are packed into the corresponding encoding character set (ASCII or EBCDIC) characters. If the second parameter does not have a zero length, then the first character of this parameter string is used to replace all the non-graphic/non-printable characters of the packed character string. When the second parameter string has a zero length, then the character "`?`" is used as the replacement character for non-graphic/non-printable characters in the return string.

  If the first parameter string does not match the regular expression then the string is considered to already be packed. In this case, the string is still checked if the second parameter length is greater than one and the non-graphic/non-printable characters are replaced by the first character of the second parameter string. When the second parameter string has a zero length, then the character "`?`" is used as the replacement character for non-graphic/non-printable characters in the return string.
- **Examples**

- `condpack('X"414141"',"?")` on an ASCII based machine returns the string `AAA`.
- `condpack('X"4141410000"',"?")` on an ASCII based machine returns the string `AAA??`.
- `condpack("4141410000","?")` on an ASCII or EBCDIC based machine returns the string `4141410000`.

A.3.16 TermAppStructDataGet, sfget

- **Synopsis**
  - `TermAppStructDataGet(String,String)`
  - `sfget(String,String)`
- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: String.
  - Return type: String.
- **Description** This funtion takes as the first parameter a value that should contain a `TermApp DE48-F0.16` Structured Data field and as the second parameter the name of a field within the structured data. The funtion will return the value of the named field as a string, if the name could not be found an empty string is returned.
- **Examples**
  - `sfget(DE48_FIELD,"OSVer")`
    Where**DE48_FIELD=**
    `219Postilion::MetaData275211FWSerialNbr11115SWRel111 19CommsType11118TermType11115OSVer11116SWHash111211F 01E201WSerialNbr2210100010000000102242315SWRel21314 4060219CommsType214INTERNAL MODEM 18TermType18EFTsma rt15OSVer19820036078 16SWHash18B4E1963A`

    returns the string `820036078`

A.3.17 TermAppStructDataSet, sfset

- **Synopsis**
  - `TermAppStructDataSet(String,String,String)`

- `sfset(String,String,String)`

- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: String.
  - Parameter 3 type: String.
  - Return type: String.
- **Description**
- **Examples**
  - ```
    sfset(DE48_FIELD,"FWSerialNbr",
          "+----------LongerValue----------+")
    ```

    Where **DE48_FIELD** is initially set to
    ```
    219Postilion::MetaData275211FWSerialNbr11115SWRel111
    19CommsType11118TermType11115OSVer11116SWHash111211F
    WSerialNbr2210100010000000100224231 5SWRel2131401E201
    4060219CommsType214INTERNAL MODEM18TermType18EFTsmar
    t15OSVer1982003607816SWHash18B4E1963A
    ```

    Will return the updated value of **DE48_FIELD** as
    ```
    219Postilion::MetaData275211FWSerialNbr11115SWRel111
    19CommsType11118TermType11115OSVer11116SWHash111211F
    WSerialNbr233+----------LongerValue----------+15SWRe
    l2131401E2014060219CommsType214INTERNAL MODEM18TermT
    ype18EFTsmart15OSVer1982003607816SWHash18B4E1963A
    ```

A.3.18 gsub, replace

- **Synopsis**
  - `gsub(String,String,String,String)`
  - `replace(String,String,String,String)`
- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: String.
  - Parameter 3 type: String.
  - Parameter 4 type: String.
    ```

- Return type: String.

- **Description** The function gsub() operates in much the same why as the `awk gsub` function does. The four parameters are

  1. Regular Expression (r) This parameter is a regular expression that should match one or more portions of the input text (t).
  2. Substitution String (s) This parameter is the replacement string
  3. Text to operate on (t) This parameter is the original input text value.
  4. How to operate (h) This parameter determines how many times the replacement text is substituted.

  How (h) can be either

  - g or G which means replace all occurrences of matched text with the substitution string.
  - Numeric which means replace only that occurrence.

  The regular expression (r) matches none, one or more portions of the input text (t) and based on the value of how (h) gsub() returns the input string where one or all of the matches are replaced with the substitution string (s).

- **Examples**

  - `gsub("a","bb",textfield,how)` This example specifies to replace the letter `a` with two letter `b`'s in `textfield` under the control of the variable `how`.

Textfield value	How	Returned Value	Description
abcdea12345a	G	bbbcdebb12345bb	Each a is replaced by two b's.
abcdea12345a	2	abcdebb12345a	The second a is replaced by two b's.
abcdea12345a	1	bbbcdea12345a	The first a is replaced by two b's.

Table 2: Effect of using gsub() to substitute text

  - `gsub("\([^ ]\+\) \([^ ]\+\)","\2 \1",textfield,how)` This example specifies to match two subtrings that contain any character except a space and that the first substring must be followed by a space followed by the second substring. The substitution string specifies to replace the whole matched value with the second matched substring followed by a space followed by the first matched substring. In other words is swaps two substrings around where the substrings do not contain a space and are separated by one space. The number of times the replacement is done is goverened by the value of the variable `how`.

Textfield value	How	Returned Value	Description
ABC DEF	G	DEF ABC	The order of the two strings is reversed.
A1 bA1 A2 BA2	G	bA1 A1 BA2 A2	Each set of two strings are reversed.
A1 bA1 A2 BA2	2	A1 bA1 BA2 A2	Only the second set is reversed.

Table 3: Effect of using gsub() to substitute text

A.3.19 alias, lookup

- **Synopsis**
  - `alias(String,String)`
  - `lookup(String,String)`
- **Parameters**
  - Parameter 1 type: String.
  - Parameter 2 type: String.
  - Return type: String.
- **Description** This function uses the second parameter as a lookup key to extract the associated value in the first parameter, which holds keyword value pairs. The value corresponding to the matched key word is returned. The keyword value pairs specified in the first parameter can either be a comma or semi-colon list of `keyword=value` pairs or a file name containing one `keyword=value` pair per line.
- **Examples**
  - `lookup("A=Alsation,L=Labrador,S=Spaniel","L")`
    Will retrun the string "Labrador"
  - `lookup("D:/lookup.txt","L")`
    will return the string "Labrador" if the file `D:/lookup.txt` holds the following:

    ```
    A=Alsation
    L=Labrador
    S=Spaniel
    ```

References

[1] objtypes: Configuring for Object Recognition, Generation and Manipulation. CML Document CML00018-01, Code Magus Limited, July 2008. PDF.

[2] expeval: Expression Evaluation User Guide. CML Document CML00091-01, Code Magus Limited, January 2013. PDF.