M-Command Manual

NYSOL Package Current Release: Ver. 2.0

Releases:
October 6, 2014: Ver. 2.0 Release
April 30, 2014: Updated errors in examples
March 10, 2014: Changes in installation method corresponding to integration of NYSOL package
February 15, 2014: Added mxml2csv
November 2, 2013: Added mpadding, mvnullo, mvdelnull, and minor changes
September 20, 2013: First Release

March 24, 2015
Copyright ©2013-2014 by NYSOL CORPORATION
Contents

1 Changes from Package 1.0 7
  1.1 Addition of Automatic Sorting Function 8
  1.2 Commands with Change in Specifications 8
  1.3 Executing Previous Scripts 9

2 M-Command 11
  2.1 M-Command 12
  2.2 Let's start 12
  2.3 Installation 14
  2.4 CSV 15
  2.5 Data Type 18
  2.6 Specify Fields 19
  2.7 Working with Data Without Field Names 24
  2.8 Multibyte Characters 25
  2.9 Specify Parameters 27
  2.10 Environment Variable 39
  2.11 Key Break Processing 41

3 Command Reference 43
  3.1 Format of MCMD 44
  3.2 maccum Cumulative Calculation 45
  3.3 marff2csv - Conversion from arff to csv Format 47
  3.4 mavg - Calculate Average 49
  3.5 mbest - Select Rows 51
  3.6 mbucket - Partition Data into Uniform Buckets 53
  3.7 mcat - ConCATenate 56
  3.8 mbucket - Multi-dimensional Uniform Bucket Partition 59
  3.9 mhashavg - Compute Average with Hash Function 61
  3.10 mhashsum - Compute Total Sum Using Hash 64
  3.11 mcombi - Compute Combination 67
  3.12 mcommon - Select Common Records in Reference File 69
  3.13 mcount - Count the Number of Rows 71
  3.14 mcross - Croosstab 72
  3.15 mcsv2arff - Convert csv to arff Format 74
  3.16 mcut - Select Column 76
  3.17 mdata - Generate Datasets 78
  3.18 mdelnull - Remove rows with null values 80
  3.19 mdformat Extract Date Time 82
  3.20 mduprec - Duplicate Record 84
  3.21 mkeybreak Keybreak Point 85
  3.22 mksort - Sort Field 87
  3.23 mhashavg - Compute Average with Hash Function 88
  3.24 mhashsum - Compute Total Sum Using Hash 90
  3.25 mkeybreak Keybreak Point 93
  3.26 mjoin - Join field(s) from Reference File 94
  3.27 mmbucket - Multi-dimensional Uniform Bucket Partition 96
  3.28 mmvavg - Calculate Moving Average 101
4 meal

4.1 meal - Computation Between Columns .................................................. 206
4.2 Components in the expression ............................................................. 207
4.3 Constant ............................................................................................... 207
4.4 Field value ............................................................................................ 207
4.5 Wildcard ............................................................................................... 207
4.6 Value from Previous Row ....................................................................... 208
4.7 Values from Next Row ............................................................................ 208
4.8 Arithmetic Operators ............................................................................. 209
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>Absolute value</td>
<td>214</td>
</tr>
<tr>
<td>acos</td>
<td>Inverse Cosine (arccosine)</td>
<td>215</td>
</tr>
<tr>
<td>age</td>
<td>Age</td>
<td>216</td>
</tr>
<tr>
<td>and</td>
<td>Logical Conjunction</td>
<td>217</td>
</tr>
<tr>
<td>argsize</td>
<td>Number of Arguments</td>
<td>218</td>
</tr>
<tr>
<td>asin</td>
<td>Inverse Sine</td>
<td>219</td>
</tr>
<tr>
<td>atan</td>
<td>Inverse Tangent</td>
<td>220</td>
</tr>
<tr>
<td>atan2</td>
<td>Angle of coordinates</td>
<td>221</td>
</tr>
<tr>
<td>avg</td>
<td>Average</td>
<td>222</td>
</tr>
<tr>
<td>bottom</td>
<td>Last row</td>
<td>223</td>
</tr>
<tr>
<td>capitalize</td>
<td>Capitalize first character</td>
<td>224</td>
</tr>
<tr>
<td>ceil</td>
<td>Ceiling</td>
<td>226</td>
</tr>
<tr>
<td>cos</td>
<td>Cosine</td>
<td>228</td>
</tr>
<tr>
<td>cosh</td>
<td>Hyperbolic Cosine</td>
<td>229</td>
</tr>
<tr>
<td>countnull</td>
<td>Sum total</td>
<td>230</td>
</tr>
<tr>
<td>day</td>
<td>Day</td>
<td>231</td>
</tr>
<tr>
<td>date</td>
<td>Year Month Day</td>
<td>232</td>
</tr>
<tr>
<td>degree</td>
<td>Degree</td>
<td>233</td>
</tr>
<tr>
<td>diff</td>
<td>Calculate Interval</td>
<td>234</td>
</tr>
<tr>
<td>dist</td>
<td>Distance</td>
<td>235</td>
</tr>
<tr>
<td>distgps</td>
<td>GPS Distance</td>
<td>236</td>
</tr>
<tr>
<td>dow</td>
<td>Day of week</td>
<td>237</td>
</tr>
<tr>
<td>e</td>
<td>Napier e Constant</td>
<td>239</td>
</tr>
<tr>
<td>exp</td>
<td>Exponential Function</td>
<td>240</td>
</tr>
<tr>
<td>factorial</td>
<td>Factorial</td>
<td>241</td>
</tr>
<tr>
<td>fixlen</td>
<td>Convert of Fixed Length String</td>
<td>242</td>
</tr>
<tr>
<td>ildsize</td>
<td>Number of Fields</td>
<td>243</td>
</tr>
<tr>
<td>floor</td>
<td>Rounding Down</td>
<td>244</td>
</tr>
<tr>
<td>format</td>
<td>Formatted Output</td>
<td>246</td>
</tr>
<tr>
<td>fract</td>
<td>Fractional Part</td>
<td>247</td>
</tr>
<tr>
<td>gcd</td>
<td>Greatest Common Divisor</td>
<td>248</td>
</tr>
<tr>
<td>hasspace</td>
<td>Search Whitespace Characters</td>
<td>249</td>
</tr>
<tr>
<td>heron</td>
<td>Area of Triangle</td>
<td>250</td>
</tr>
<tr>
<td>hour</td>
<td>Hour</td>
<td>251</td>
</tr>
<tr>
<td>if</td>
<td>Conditional Statements</td>
<td>252</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
<td>254</td>
</tr>
<tr>
<td>match</td>
<td>Search</td>
<td>268</td>
</tr>
<tr>
<td>isnull</td>
<td>Evaluate NULL value</td>
<td>257</td>
</tr>
<tr>
<td>julian</td>
<td>Julian Calendar Conversion</td>
<td>258</td>
</tr>
<tr>
<td>lcm</td>
<td>Least Common Denominator</td>
<td>259</td>
</tr>
<tr>
<td>leapyear</td>
<td>Determine Leap Year</td>
<td>260</td>
</tr>
<tr>
<td>left</td>
<td>Extract String from Left</td>
<td>261</td>
</tr>
<tr>
<td>length</td>
<td>Length of Character String</td>
<td>262</td>
</tr>
<tr>
<td>line</td>
<td>Line Number</td>
<td>263</td>
</tr>
<tr>
<td>ln</td>
<td>Natural Logarithm</td>
<td>264</td>
</tr>
<tr>
<td>log</td>
<td>Logarithm</td>
<td>265</td>
</tr>
<tr>
<td>log10</td>
<td>Common logarithm</td>
<td>266</td>
</tr>
<tr>
<td>log2</td>
<td>Log base 2</td>
<td>267</td>
</tr>
<tr>
<td>match</td>
<td>Search</td>
<td>268</td>
</tr>
<tr>
<td>max</td>
<td>Maximum Value</td>
<td>270</td>
</tr>
<tr>
<td>mid</td>
<td>Extract Substring</td>
<td>271</td>
</tr>
<tr>
<td>min</td>
<td>Minimum Value</td>
<td>272</td>
</tr>
<tr>
<td>minute</td>
<td>Minute</td>
<td>273</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>4.68</td>
<td>month - Month</td>
<td>274</td>
</tr>
<tr>
<td>4.69</td>
<td>not - Negation</td>
<td>276</td>
</tr>
<tr>
<td>4.70</td>
<td>now - Current Time</td>
<td>277</td>
</tr>
<tr>
<td>4.71</td>
<td>null - NULL Value</td>
<td>278</td>
</tr>
<tr>
<td>4.72</td>
<td>nrand - Random Numbers in Normal Distribution</td>
<td>279</td>
</tr>
<tr>
<td>4.73</td>
<td>or - Logical Disjunction</td>
<td>280</td>
</tr>
<tr>
<td>4.74</td>
<td>pi - Pi</td>
<td>281</td>
</tr>
<tr>
<td>4.75</td>
<td>power - Raised to a Power</td>
<td>282</td>
</tr>
<tr>
<td>4.76</td>
<td>product - Product</td>
<td>283</td>
</tr>
<tr>
<td>4.77</td>
<td>radian - Radian</td>
<td>284</td>
</tr>
<tr>
<td>4.78</td>
<td>rand - Uniformly distributed random numbers</td>
<td>285</td>
</tr>
<tr>
<td>4.79</td>
<td>randi - Uniformly Generated Random Integers</td>
<td>286</td>
</tr>
<tr>
<td>4.80</td>
<td>regexlen - Match Length of Character String</td>
<td>287</td>
</tr>
<tr>
<td>4.81</td>
<td>regexm - Complete Match</td>
<td>289</td>
</tr>
<tr>
<td>4.82</td>
<td>regexpfx - Prefix of Match String</td>
<td>290</td>
</tr>
<tr>
<td>4.83</td>
<td>regexpos - Match Position</td>
<td>291</td>
</tr>
<tr>
<td>4.84</td>
<td>regexrep - Replace Match String</td>
<td>293</td>
</tr>
<tr>
<td>4.85</td>
<td>regexs - Substring Match</td>
<td>294</td>
</tr>
<tr>
<td>4.86</td>
<td>regexsfx - Suffix of Matched String</td>
<td>295</td>
</tr>
<tr>
<td>4.87</td>
<td>regexpr - Character String</td>
<td>296</td>
</tr>
<tr>
<td>4.88</td>
<td>right - Extract Substring from Right</td>
<td>297</td>
</tr>
<tr>
<td>4.89</td>
<td>round - Rounding</td>
<td>298</td>
</tr>
<tr>
<td>4.90</td>
<td>second - Second</td>
<td>300</td>
</tr>
<tr>
<td>4.91</td>
<td>sign - Sign</td>
<td>301</td>
</tr>
<tr>
<td>4.92</td>
<td>sin - Sine</td>
<td>302</td>
</tr>
<tr>
<td>4.93</td>
<td>sinh - Hyperbolic Sine</td>
<td>303</td>
</tr>
<tr>
<td>4.94</td>
<td>sqrt - Square Root</td>
<td>304</td>
</tr>
<tr>
<td>4.95</td>
<td>sqsum - Sum of Squares</td>
<td>305</td>
</tr>
<tr>
<td>4.96</td>
<td>sum - Sum</td>
<td>306</td>
</tr>
<tr>
<td>4.97</td>
<td>tan - Tangent</td>
<td>307</td>
</tr>
<tr>
<td>4.98</td>
<td>tanh - Hyperbolic Tangent</td>
<td>308</td>
</tr>
<tr>
<td>4.99</td>
<td>time - Hour Minute Second</td>
<td>309</td>
</tr>
<tr>
<td>4.100</td>
<td>today - Today's date</td>
<td>310</td>
</tr>
<tr>
<td>4.101</td>
<td>tolower - Lowercase Conversion</td>
<td>311</td>
</tr>
<tr>
<td>4.102</td>
<td>top - Top Rows</td>
<td>312</td>
</tr>
<tr>
<td>4.103</td>
<td>toupper - Uppercase Conversion</td>
<td>313</td>
</tr>
<tr>
<td>4.104</td>
<td>tseconds - Elapsed Time in Seconds</td>
<td>314</td>
</tr>
<tr>
<td>4.105</td>
<td>uxt - UNIX Time Conversion</td>
<td>315</td>
</tr>
<tr>
<td>4.106</td>
<td>Week</td>
<td>316</td>
</tr>
<tr>
<td>4.107</td>
<td>year - Liturgical Year</td>
<td>317</td>
</tr>
<tr>
<td>4.108</td>
<td>Cast</td>
<td>318</td>
</tr>
</tbody>
</table>
Chapter 1

Changes from Package 1.0
1.1 Addition of Automatic Sorting Function

In MCMD Ver. 1.0, sorting by key field with msortf is necessary prior to the aggregate calculation with commands such as msum and joining of files such as mjoin command.

Key fields must be sorted for commands which require specification key field name, or else it would result in calculation error if users failed to sort the key field beforehand, this is the main draw back easily give rise to bugs in scripts.

Therefore, the new function in Ver. 2.0 automatically sorts key columns for commands where key field(s) is/are specified at k= option. The symbol of sort order by column is added to the field name in the CSV header (Refer to sort Sort order of columns for more information), and sorting is only carried out when necessary on required commands.

Example

Example of Ver. 1.0

Before executing msum k=customer command, customer column must be sorted with msortf command.

$ more dat1.csv
customer,amount
A,10
B,10
A,20
B,15
B,20
$ msortf i=dat1.csv f=customer | msum k=customer f=amount:totalamount o=rsl1.csv
#END# kgsortf f=customer i=dat1.csv
#END# kgsum f=amount:totalamount k=customer o=rsl1.csv
$ more rsl1.csv
customer,totalamount
A,30
B,45

Example of Ver. 2.0

msum will perform sorting when necessary without the use of msortf command while achieving the correct results as shown in the previous example. The symbol %0 is added next to the column name of customer after sorting is carried out in msum command.

$ more dat1.csv
customer,amount
A,10
B,10
A,20
B,15
B,20
$ msum i=dat1.csv k=customer f=amount:totalamount o=rsl1.csv
#END# kgsum f=amount:totalamount k=customer i=dat1.csv o=rsl1.csv
$ more rsl1.csv
customer%0,totalamount
A,30
B,45

1.2 Commands with Change in Specifications

Table 1.1 shows the list of commands with specification changes from Ver. 1.0 (Excludes commands with k= parameter where automatic sorting has been added ). In addition, the s= parameter is added to the commands where the sort order of records will affect the processing results.
1.3 Executing Previous Scripts

When `-q` option is used, automatic sorting on columns defined at `k=` parameter is disabled. As shown in Table 1.1, when `s=` parameter is not defined for commands that takes in `s=` parameter, the commands will operate as in Ver. 1.0. For scripts using Ver. 2.0 commands where `k=` parameter and `-q` option is specified, while `s=` parameter is not defined, the results is equivalent to scripts using Ver. 1.0 commands (For `mpadding` command, even though `type=` parameter is removed, the specification is required at `f=` parameter).

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>maccum</td>
<td>Calculate cumulative totals</td>
<td>Added <code>s=</code> parameter (required)</td>
</tr>
<tr>
<td>mbest</td>
<td>Select specific rows</td>
<td>Added <code>s=</code> parameter (required)</td>
</tr>
<tr>
<td>mcomb</td>
<td>Compute combinations</td>
<td>Added <code>s=</code> parameter</td>
</tr>
<tr>
<td>mkeybreak</td>
<td>Keybreak point</td>
<td>Added <code>s=</code> parameter</td>
</tr>
<tr>
<td>mmvavg</td>
<td>Compute moving average</td>
<td>Added <code>s=</code> parameter (required)</td>
</tr>
<tr>
<td>mmvssim</td>
<td>Compute similarity of sliding window</td>
<td>Added <code>s=</code> parameter (required)</td>
</tr>
<tr>
<td>mmvstats</td>
<td>Compute statistics of sliding window</td>
<td>Added <code>s=</code> parameter (required)</td>
</tr>
<tr>
<td>mnumber</td>
<td>Serial number</td>
<td><code>s=</code> parameter is required when <code>-B</code> is not used.</td>
</tr>
<tr>
<td>mpadding</td>
<td>Repair rows</td>
<td><code>type=</code> parameter is not retired, repair method is specified at <code>f=</code>.</td>
</tr>
<tr>
<td>mrjoin</td>
<td>Join reference file by specified range</td>
<td>Specify sorting method at <code>r=</code> parameter.</td>
</tr>
<tr>
<td>mslide</td>
<td>Shift records</td>
<td>Added <code>s=</code> parameter (required)</td>
</tr>
<tr>
<td>mtra</td>
<td>Convert vert data to vector</td>
<td>Added <code>s=</code> parameter</td>
</tr>
<tr>
<td>mwindow</td>
<td>Generate sliding window</td>
<td>Specify sorting method at <code>wk=</code>parameter.</td>
</tr>
</tbody>
</table>

Example on Script Modifications

Script Before Modification

```
msortf i=customer.csv f=custID,date |
maccum k=custID f=amount o=accum.csv
```

#ERROR# parameter `s=` is mandatory without `-q` (`kgaccum`); `kgaccum f=amount i=test.csv k=custID`

Modification

`s=` parameter is required for `maccum` command, when `-q` option is used, it is equivalent to the execution method in Ver. 1.0.

```
msortf i=customer.csv f=custID,date |
maccum -q k=custID f=amount o=accum.csv
```
Chapter 2

M-Command
CHAPTER 2. M-COMMAND

2.1 M-Command

M-Command (also referred to as MCMD) is a set of commands that are developed for the purpose of high-speed processing (CSV) of large-scale structured data tables. Its origin can be traced back to the early 1990s. Mr. Yasuyuki Matsuda invented this data processing methodology and commands for large-scale system development projects which was implemented in big companies. The M in M-Command refers to Mr. Matsuda’s initial.

Mr. Matsuda’s invention is not just a technology breakthrough, it fundamentally examines how information system can be customized for business use. The system is said to be completely different system design concept then the West. The ideological concept of M command is explained in another paper, nevertheless, it is synonymous with current big data processing technology.

M-Command provides about 70 different commands where each command is specific to a single function (For example, sort or join tables). The basic tenets of what all commands have in common is it reads CSV data from standard input, and carry out a very simple processing method and write the results to standard output. It achieves a variety of processing functions by connecting individual commands with an inter-process stream called "pipe", thus the output of each process feeds directly as input to the next one. These features are the same as the UNIX philosophy. With M-Command, an information system can be created based solely using M-Command and UNIX utilities on a UNIX platform.

It is possible to efficiently process large scale data with hundred millions row of records on a standard PC with M-Command using a standard PC, and it does not require a lot of time to learn M-Command. The command has been included in the curriculum for undergraduate Liberal Arts students in the past, after a few weeks of practice, you will be able to master the commands for data processing.

2.2 Let’s start

Let’s start with a simple example. If you have already installed MCMD, confirm the following example by typing the same at the command line. Sample data is required in order to proceed with the tutorial. For MCMD, mdata command can be used to create different kinds of data set, refer to the example in Figure 2.1. The line beginning with $ indicates the command line input, followed by the execution message and results from the command. The mdata command reads the data type from the parameter and directs the contents to standard output (Refer to chapter on mdata for more details). In the following example, the data is saved to the file man0.csv by redirecting the contents to the standard output.

After MCMD command is executed successfully, it returns a status message starting with #END#. In addition, more is a UNIX command to display the contents of the file one page at a time which allows forward navigation of the file. All examples in this manual are illustrated according to the above-mentioned format.

```bash
$ mdata man0 >man0.csv
#END# mdata man0
$ more man0.csv
顧客, 金額
A,5200
B,4000
B,3500
A,2000
B,800
```

Figure 2.1: Generate data with mdata command

Using the data man0.csv generated in Figure 2.1, the following example in Figure 2.2 calculates the total amount for each customer.

Input data is a CSV text file consisting of five records, and each record consist of two fields namely "customer" and "amount". Sort the data according to customer using msortf command, and pass the result to msum command by connecting the data stream with pipe(|) pipe. At the msum command, the customer field is set as the key field to calculate the total amount and the result are written to the output file output.csv. M-Command displays a status message when it terminates the process. If the command executes successfully,
the output message starts with #END#, however, if the commands terminates with error, the message will start with #ERROR#.

```bash
$ msortf f=customer i=man0.csv | msum k=customer f=amount o=output.csv
#END# kgsortf f=customer i=man0.csv
#END# kgsum f=amount k=customer o=output.csv
$ more output.csv
customer,amount
A,7200
B,9300
```

Figure 2.2: Total amount by each customer

Let's explore a more complex example. The example shown in Figure 2.3, displays the product and corresponding quantity by each customer in a matrix format. Instead of using pipe, the results of each command is written to an output file, and the content of the output file is displayed with more command. Line comments begin with # #.

The function of mcut command only extract the specified field, the mcount command counts the number of rows, and mcross command performs cross tabulation on the data.

Rather than dictating detailed processes of each command, the following example shows how the input data is processed for each command. M-Command consist of over 70 commands, a variety of data processing can be carried out by flexibly by combining multiple commands.

```bash
$ mdata man1 >man1.csv
#END# mdata man1
$ more man1.csv
顧客，日付，商品
A,20130916,a
A,20130916,c
A,20130917,a
A,20130917,e
B,20130916,d
B,20130917,a
B,20130917,d
B,20130917,f
$ mcut f=customer,product i=man1.csv o=xxa
#ERROR# field name not found: 'customer' in man1.csv (kgcut)
$ more xxa
$ msortf f=customer,product i=xxa o=xxb
#ERROR# no data found : xxa (kgsortf)
$ more xxb
# Count the number of rows by customer and product.
$ mcount k=customer,s=product a=number of items i=xxb o=xxc
#ERROR# invalid argument: of (kgcount)
$ more xxc
xxc: No such file or directory
# Perform a cross tabulation by the item of product. The number of the product that is not purchased gives 0.
$ mcross k=customer s=product f=number of items v=0 i=xxc o=xxd
#ERROR# invalid argument: of (kgcross)
$ more xxd
xxd: No such file or directory
# remove extra item "fld".
$ mcut f=field -r i=xxd o=output.csv
#ERROR# file not found : xxd (kgcut)
$ more output.csv
output.csv: No such file or directory
```

Figure 2.3: Customer product purchase quantity matrix

Furthermore, information about the usage of each command is available in help (Table 2.4) by specifying the --help option. Use --version parameter to display the current version of MCMD. Note that version displays the version of MCMD, thus the same version will be displayed in all individual commands.
### mcut Selection of field

Select the specified field. If you give the -r option, the specified field is deleted.

**Format**

```
mcut [-r] [i=] [o=] [-nfn] [-nfno] [-x] [--help] [--version]
```

**Parameters**

- `f=`: Extract field name
  - By separating by a colon the field name, it is possible to change the output field name. ex. `f=a:A,b:B`
- `-r`: Field removal switch
  - Delete the fields specified by `f=`, and other fields are extracted.
- `-nfn`: The first line of input data is not considered as field header.
  - Therefore, it must be specified by the field number.
  - Output field name can be added by specifying a combination of new field name.
  - Example) `f = 0: date, 2: shop, 3 Quantity`

```bash
$ mcut --version
lib Version 1:1:0:0
```

Figure 2.4: Display help information

## 2.3 Installation

This package is part of the complete nysol package. Prerequisite software is required before the installation of nysol software package. More details are available in nysol package installation instructions ([http://www.nysol.jp/en/home/install](http://www.nysol.jp/en/home/install)).
2.4 CSV

MCMD processes tabular data in CSV format (Comma Separated Values) as illustrated in Figure 2.5. CSV is a de facto standard format for table-structured data. It is widely used as a tabular format to import and export data between application programs.

![Example of CSV data](image)

However, CSV is not a standard format endorsed by organization for standardization nor corporate initiatives, as a result the method of handling a CSV differs from each software vendor at present. The proposed RFC 4180 is an effort to formalize CSV as an Internet standard in October 2005 is a significant move to increase the portability of CSV. Augmented Backus-Naur Format (ABNF) for CSV files in RFC 4180 is shown in Figure 2.6.

![Definition of ABNF for CSV](image)

Meaning of each line in Figure 2.6 is as follows.

- (A) File consists of a header and record of one or more lines. Header is not required. The line break (CRLF) is attached at the end of the header and each record. The line break (CRLF) in the last row is not required.
- (B) Header consists of one or more names which is separated by a single comma.
- (C) Record consists of one or more fields which is separated by a single comma.
- (D) Name refers to field.
- (E) Field can include an escape character or non-escape character.
- (F) Field values containing 1 or more text characters (TEXTDATA), comma(COMMA), newline character (CR or LF) shall have a pair of consecutive double quotes character escaped by doubling it.
- (G) Non-escape refers to 1 or more text characters (TEXTDATA).
- (H) ASCII code of comma in hexadecimal is 2C.
- (I) ASCII code of carriage return (CR) in hexadecimal is 0D.
- (J) ASCII code of double quotation (DQUOTE) in hexadecimal is 22.
- (K) ASCII code of line feed (LF) in hexadecimal is 0A.
- (L) Line break or newline is represented as a carriage return + line feed.
- (M) Text character (TEXTDATA) had the range of 20-21, 23-2B, 2D-7E in hexadecimal ASCII code.
2.4.1 Definition specific to KGMOD

KGMOD (MCMD) added the following rules to CSV as defined above.

- The number of fields must be the same in all the rows.
- Set a limit on the maximum length of a single row (default value is 1024000 bytes (1MB) and expandable up to 10MB).
- Line break is only marked with Line Feed (LF).
- Line break is mandatory even in the last record.
- Added the 80-FF (hexadecimal) range to text characters for handling multibyte characters.

It is sufficient to use the `mchkcsv` command to verify whether the CSV file meets the above definition.

2.4.2 Common input and output process

The input and output sequence of CSV file for MCMD follows the steps listed below.

1. Read file into memory blocks.
2. Split the comma-delimited string into different fields while taking consideration of escape character.
3. Interpret escape characters and convert to original data (except DQUOTE).
4. Run the specific processing function of the command and write the results to the output buffer.
5. Add character escapes if necessary.
6. Output to a file when buffer is full.

2.4.3 Notes

For points to note when preparing the CSV data will be described below with examples.

Data containing comma characters

Escape comma characters in data by enclosing them in double quotes. The following is a CSV file comprising of two fields `f1,f2`. The data in row 0\(^1\) at column `f1` is enclosed in double quotes since it contains a comma.

```
f1,f2
"abc,def",2
xyz,2
```

Data containing double quotes

Data containing double quotes characters can be represented by a pair of consecutive double quote. The following is the CSV file that consists of two columns `f1,f2`. Data in row 0 and 1 at column `f1` is escaped with double quotation. The original data is written as `abc"def` and `"` respectively.

```
f1,f2
"abc""def",2
"",2
```

\(^1\)MCMD address the value of the first row as 0 (except for the field name row) consistently.
2.4. CSV

Line breaks in data

Data including a line break can be processed when enclosed in double quotes. A line break is included in the data at row 0 in column f1 after abc, since the data is enclosed in double quotes, it is identified as part of the data instead of end of the line.

<table>
<thead>
<tr>
<th>f1,f2</th>
</tr>
</thead>
</table>
| "abc  
def",1 |

Unnecessary double quotes

Double quotes in data are removed in the output where unnecessary.

$ more data.csv
f1,f2
"abc",efg
abc,"efg"

$ mcut f=f1,f2 i=data.csv
f1,f2
abc,efg
abc,efg
2.5 Data Type

MCMD handles plain text files in CSV format, where the data is a sequence of characters. Thus, it depends on how the specific command interprets the character string for the data type. For example, the data in field specified at $f_*$ in $\textit{msum}$ command is converted from a character string to a number. As shown in Table 2.1, MCMD can handle six types of data including numeric, character string, date, time, boolean and vector type.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Notation of CSV Data type</th>
<th>Details of Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical type</td>
<td>10, 2.5, 1.5E+10</td>
<td>Convert value to a double-precision real number</td>
</tr>
<tr>
<td>Character string type</td>
<td>abc, あいう</td>
<td>Process CSV data as it is</td>
</tr>
<tr>
<td>Date type</td>
<td>20130920</td>
<td>Convert 8-digit fixed length string to Gregorian calendar object</td>
</tr>
<tr>
<td>Time type</td>
<td>20130920151154, 151154</td>
<td>Convert 6 digit or the 14-digit fixed length string to Gregorian calendar + POSIX time object</td>
</tr>
<tr>
<td>Boolean type</td>
<td>1,0</td>
<td>Convert character to boolean value. &quot;1&quot; is true and 0 is false</td>
</tr>
<tr>
<td>Vector type</td>
<td>a c b, 1 5 11</td>
<td>Character string delimited by space can be converted to any data type above</td>
</tr>
</tbody>
</table>

Further, list of data types of commonly used commands is shown in Table 2.2.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Command</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical type</td>
<td>$\textit{msum}$</td>
<td>Calculate total of numeric field</td>
</tr>
<tr>
<td></td>
<td>$\textit{msim}$</td>
<td>Calculate the similarity between two fields</td>
</tr>
<tr>
<td>String type</td>
<td>$\textit{mjoin}$</td>
<td>Combine fields from the reference file</td>
</tr>
<tr>
<td></td>
<td>$\textit{mcombi}$</td>
<td>Enumerate combination</td>
</tr>
<tr>
<td>Date type</td>
<td>$\text{age}$ function of $\textit{mcal}$</td>
<td>Calculate Age</td>
</tr>
<tr>
<td></td>
<td>$\text{leapyear}$ function of $\textit{mcal}$</td>
<td>Determine leap year</td>
</tr>
<tr>
<td>Time type</td>
<td>$\text{now}$ function of $\textit{mcal}$</td>
<td>Output the current time</td>
</tr>
<tr>
<td></td>
<td>$\text{diffminute}$ function of $\textit{mcal}$</td>
<td>Calculate the time difference in minutes</td>
</tr>
<tr>
<td>Boolean type</td>
<td>$\text{and}$ function of $\textit{mcal}$</td>
<td>Compute the logical product</td>
</tr>
<tr>
<td></td>
<td>$\text{if}$ function of $\textit{mcal}$</td>
<td>Set the value of the criteria</td>
</tr>
<tr>
<td>ベクトル型</td>
<td>$\textit{mvsort}$</td>
<td>Sort vector elements</td>
</tr>
<tr>
<td></td>
<td>$\textit{mvuniq}$</td>
<td>Extract unique vector elements</td>
</tr>
</tbody>
</table>
2.6 Specify Fields

MCMD reads the field names in the first row of CSV data, the field can also be specified with a field number without the field names. There are four options for handling the row of field names includes -nfn,-nfno,-nfni and -x. Its usage will be illustrated with examples as follows. In addition, note that the field number starts at 0 from the left such as 0, 1, 2.

Example 1: Specify -nfn

When -nfn (no field name) is specified, the first row in the data will not be considered as field names. Thus, each field is specified as a number (note that the number starts from 0).

```
$ more dat2.csv
a,2
b,5
b,4
$ msum -nfn k=0 f=1 i=dat2.csv o=rsl1.csv
#END# kgsum -nfn f=1 i=dat2.csv k=0 o=rsl1.csv
$ more rsl1.csv
a,2
b,9
```

Example 2: Specify -nfno

When -nfno (no field name for output) is specified, the first row of input data is initialized as field names, but the field names is removed from the output data.

```
$ more dat1.csv
key,val
a,2
b,5
b,4
$ msum k=key f=val -nfno i=dat1.csv o=rsl2.csv
#END# kgsum -nfno f=val i=dat1.csv k=key o=rsl2.csv
$ more rsl2.csv
a,2
b,9
```

Example 3: Specify -nfni

The option -nfni (no field name for input) is only available in the mcut command. This option does the opposite of -nfno, where the first row of input data is treated as data items, but the field names will be shown in the output data.

```
$ mcut f=0:val -nfni i=dat2.csv o=rsl3.csv
#END# kgcut -nfni f=0:val i=dat2.csv o=rsl3.csv
$ more rsl3.csv
key,val
a,2
b,5
b,4
```

Example 4: Specify -x

For CSV data with a field names, use the -x option to specify the field number.

```
$ msum -x k=0 f=1 i=dat1.csv o=rsl4.csv
#END# kgsum -x f=1 i=dat1.csv k=0 o=rsl4.csv
$ more rsl4.csv
key,val
a,2
b,9
```
2.6.1 Valid field names

Field name can contain the characters stated as follows:

- Multibyte characters (such as UTF-8)
- Alphabet (a-z, A-Z)
- Number (0-9)
- Symbol

However, it is recommended to avoid using the following symbols. Using the symbols will not return an error, however, the symbol maybe be used as special characters in MCMD and the special character function may not be available if it is used in as a field name.

- , Comma
- : Colon
- % Percent
- * Asterisk
- ? Question mark
- & And
- \ Backslash
- ] Square brackets, right
- [ Square brackets, left

2.6.2 Valid item number

When specifying field number, the field numbers can be listed with a comma delimiter. Alternatively, it is also possible to specify the number at the end of the field name (add "L") or specify the range (-).

For example, the argument 0L specifies the last field, and 2L specifies the 2nd field from the end (note that field number starts from 0). Furthermore, when 0-5 is specified, six fields starting from 0 to 5 are selected, which is equivalent to 0,1,2,3,4,5

Example 1: Specify range

By specifying "0-4", fields 0,1,2,3,4 are specified.

```
$ more dat1.csv
brand,quantity01,quantity02,quantity03,quantity04,quantity05,quantity06,quantity07,quantity08,quantity09,quantity10
A,10,50,90,130,170,210,250,290,330,370
B,20,60,100,140,180,220,260,300,340,380
C,30,70,110,150,190,230,270,310,350,390
D,40,80,120,160,200,240,280,320,360,400
$ mcut -x f=0-4 i=dat1.csv o=rsl1.csv
#END# kgcut -x f=0-4 i=dat1.csv o=rsl1.csv
$ more rsl1.csv
brand,quantity01,quantity02,quantity03,quantity04
A,10,50,90,130
B,20,60,100,140
C,30,70,110,150
D,40,80,120,160
```

Example 2: Specify range in reverse order

By specifying ⬇️ 4-0 ⬆️, fields 0,1,2,3,4 are specified.

```
$ mcut -x f=4-0 i=dat1.csv o=rsl1.csv
#END# kgcut -x f=4-0 i=dat1.csv o=rsl1.csv
$ more rsl1.csv
brand,quantity01,quantity02,quantity03,quantity04
A,10,50,90,130
B,20,60,100,140
C,30,70,110,150
D,40,80,120,160
```
2.6. SPECIFY FIELDS

Example 3: Specify Multiple ranges

By specifying \[1-0,2-4\], fields \[1,0,2,3,4\] are specified.

EOF scp=<<EOF
mcut -x f=1-0,2-4 i=dat1.csv o=rsl3.csv
EOF

Example 4: Specified field from the end

By specifying "2L", the second field from the end is specified (quantity 08).

Example 5: Specify the range of fields from the end

By specifying "5-3L", the 5th to the 3rd item from end is specified, i.e. "5,6,7".

2.6.3 1.6.3 Input and output fields

The \(f\) parameter is used to specify the field(s) in many commands. The format of \(f\) is defined as \(input field:output field\). If output field name is not specified, the same input field name will be used as output field name. In addition, it is also possible to specify combinations of field names such as \(f=0:quantity\).

Example 1: Basic Example

By specifying "quantity:unit sales", the field name is converted from \(quantity\) to \(unit sales\) in the output.
Example 2: Add field name

The maccum command accumulates the values in the "quantity" field, and add the field name "cumulative quantity" in the output results. If the parameter is specified as "f=quantity", the field name of the cumulative result will remain as "quantity", thus results in error because the same field name □ quantity □ exists in the output.

```
$ maccum f=quantity:accumulationquantity i=dat1.csv o=rsl2.csv
#ERROR# same field name is specified: quantity (kgaccum)
```

Example 3: Mixing field name and field number

The field name and field number can be specified at the same time.

```
$ mcut f=0,1:salesquantity -x i=dat1.csv o=rsl3.csv
#END# kgcut -x f=0,1:salesquantity i=dat1.csv o=rsl3.csv
$ more rsl3.csv
brand,salesquantity
A,10
B,20
C,30
D,40
```

2.6.4 Wildcard

The wildcard characters "*" and "?" can be used to specify multiple field names. The asterisk sign "*" matches 0 or more characters, and the question mark "?" matches a single character. Note that the order of evaluation of wildcard characters follows the order of the fields in the input data. For example, if the order of the fields in input data is A5, A3, A4, A2, A1, the parameter f=A* is evaluated as f=A5, A3, A4, A2, A1.

Example 1: Basic Example

The expression "quantity*" matches field names starting with quantity ("quantity10", "quantity11", "quantity12" and "quantity13").

```
$ more dat1.csv
brand,quantity10,quantity11,quantity12,quantity13
A,10,15,9,1
B,20,16,8,2
C,30,17,7,3
D,40,18,6,4
```
2.6. SPECIFY FIELDS

$ mcut f= quantity* i=dat1.csv o=rsl1.csv
#ERROR# invalid argument: quantity* (kgcut)
$ more rsl1.csv
rsl1.csv: No such file or directory

Example 2: Wildcard character ? ?
Select field names which begin with "quantity" followed by 1, and match any single character after 1. In this case, the wildcard does not match with field name "quantity123".

$ mcut f= quantity 1? i=dat1.csv o=rsl2.csv
#ERROR# invalid argument: quantity (kgcut)
$ more rsl2.csv
rsl2.csv: No such file or directory

2.6.5 Replace the name of an output field
The special character "&" specified in the output field name can be replaced with the current field name. For example, the parameter f=abc:xx&xx returns xxabcxx as the output field name. The "&" character can be specified at any position as many times as required in the output field name. However, the ampersand is a special character in shell which is interpreted as "background execution". Thus, it is necessary to escape and enclose the field name in double quotes when including "&" in field name.

Example 1: Basic Example
In this example, "&" is replaced with br<e>and in the input field name, which is equivalent to the expression "f=brand:brand code".

$ more dat1.csv
brand,quantity10,quantity11,quantity12,quantity123
A,10,15,9,1
B,20,16,8,2
C,30,17,7,3
D,40,18,6,4
$ mcut f="brand:& code" i=dat1.csv o=rsl1.csv
#END# kgcut f=brand:& code i=dat1.csv o=rsl1.csv
$ more rsl1.csv
brand code
A
B
C
D

Example 2: Combine with wildcard
Attach & & after sales& to replace the character with input field name (e.g. "quantity10") in the output field name. For all input fields name beginning with "quantity", attach sales as the prefix in the output field name.

$ mcut f="brand,quantity*:sales&" i=dat1.csv o=rsl2.csv
#END# kgcut f=brand,quantity*:sales& i=dat1.csv o=rsl2.csv
$ more rsl2.csv
brand,salesquantity10,salesquantity11,salesquantity12,salesquantity123
A,10,15,9,1
B,20,16,8,2
C,30,17,7,3
D,40,18,6,4
2.7 Working with Data Without Field Names

If field names are included in CSV input data, the set of field names are usually included in the output corresponding to the context of data processing.

On the other hand, if the CSV input data do not include field names, and data is 0 byte file, the result will also return 0 byte file. Both the number of input and output records are 0.

Example 1: Data with field name

```
$ more dat1.csv
A,B,C
$ msetstr v="string" a=X i=dat1.csv o=rsl1.csv
#END# kgsetstr a=X i=dat1.csv o=rsl1.csv v=string
$ more rsl1.csv
A,B,C,X
```

Example 2: Data without field name

```
$ more dat2.csv
$ msetstr v="string" -nfn i=dat2.csv o=rsl2.csv
#END# kgsetstr -nfn i=dat2.csv o=rsl2.csv v=string
$ more rsl2.csv
```

2.8 Multibyte Characters

MCMD handles multibyte characters such as Chinese characters in UTF-8 encoding. Other encodings such as SHIFT-JIS can be treated as multibyte characters, however, some functions may not work correctly. The following explains how MCMD process multibyte characters.

Kanji-code is processed as multibyte characters without conversion in order to increase the processing speed when using MCMD. However, character string search and string substitution functions may result in unexpected results depending on the encoding.

For example, “陰 (shadow)” is represented as 0x8941 in SHIFT-JIS, the second byte of this character refers to “A” in single-byte characters. Thus, when 陰 A 隱 is substituted with 隱 B 隱, “陰 隱” will be converted to 陰 隱 (hidden) 隱 (0x8942). The UTF-8 uses an encoding system which could avoid problems with character substitution. Moreover, it is difficult to count the number of characters in strings containing multibyte characters and ASCII characters even in UTF-8.

This problem can be avoided by converting all characters including ASCII code to fixed length character, known as wide character (MCMD adopts 32-bit fixed length).

When converting wide characters, it is necessary to find out the encoding for multibyte characters in the environment variable LANG. Type the following at the command prompt to check the environment variable,

```
$ echo $LANG
ja_JP.UTF-8
```

Some MCMD commands have built-in option (-W) to convert input data to wide characters before data processing. The list of commands which support the option is shown in Table 2.3. These commands pertain to search or replace functions, it is not necessary to use this option if encoding is set as UTF-8.

<table>
<thead>
<tr>
<th>Command name</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mchgstr</td>
<td>Substitution</td>
<td>-By specifying -W, the field data specified by f= is converted to wide characters internally.</td>
</tr>
<tr>
<td>mselstr</td>
<td>Search</td>
<td>In case of substring matching (-sub), the field data specified by f= is converted to wide characters internally.</td>
</tr>
<tr>
<td>msed</td>
<td>Substitution</td>
<td>By specifying -W, the field data specified by f= is converted to wide characters internally.</td>
</tr>
<tr>
<td>mtonull</td>
<td>Search</td>
<td>For substring matching (-sub), the field data specified by f= is converted to wide characters internally.</td>
</tr>
</tbody>
</table>

In addition, mcall and msel incorporated functions to handle wide characters (Table 2.4). For instance, the lengthw function counts the number of characters and computes the character position for data in UTF-8 encoding.

Take note of the following when handling wide-character.

- Conversion to wide character involves overhead which sacrifices the processing speed.
- Wide characters input data can be converted except for the field names.
- File name with multibyte characters can be processed as it.
<table>
<thead>
<tr>
<th>Name of the function</th>
<th>Function</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>lengthw</td>
<td>Number of characters</td>
<td>Convert target string to wide character before processing.</td>
</tr>
<tr>
<td>midw</td>
<td>Substring</td>
<td>Convert target string to wide character before processing.</td>
</tr>
<tr>
<td>rightw</td>
<td>Substring</td>
<td>Convert target string to wide character before processing.</td>
</tr>
<tr>
<td>leftw</td>
<td>Substring</td>
<td>Convert target string to wide character before processing.</td>
</tr>
<tr>
<td>regexsw</td>
<td>Match regular expression</td>
<td>Convert target string to wide character before processing.</td>
</tr>
<tr>
<td>regexmw</td>
<td>Match regular expression</td>
<td>Convert target string to wide character.</td>
</tr>
<tr>
<td>regexrepw</td>
<td>Substitute by regular expression</td>
<td>Convert target string to wide character.</td>
</tr>
<tr>
<td>regexlenw</td>
<td>Match length by regular expression</td>
<td>Convert target string to wide character.</td>
</tr>
<tr>
<td>regexposw</td>
<td>Match position by regular expression</td>
<td>Convert target string to wide character.</td>
</tr>
<tr>
<td>regexstrw</td>
<td>Substring match by regular expression</td>
<td>Convert target string to wide character.</td>
</tr>
<tr>
<td>regexpfw</td>
<td>Prefix by regular expression</td>
<td>Convert target string to wide character.</td>
</tr>
<tr>
<td>regexsfw</td>
<td>Suffix match by regular expression</td>
<td>Convert target string to wide character.</td>
</tr>
</tbody>
</table>
2.9 Specify Parameters

The format of the parameters used in M-Command is slightly different than UNIX commands. The keyword and specified value is separated by an equal sign i.e. "keyword=value". Option type parameters precedes with a minus sign e.g. "-keyword" and do not require specified value.

Many parameters share common functions in M-Command. The parameters are explained below. However, in some command, it works as a completely different function.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=</td>
<td>Input file name</td>
</tr>
<tr>
<td>o=</td>
<td>Output file name</td>
</tr>
<tr>
<td>f=</td>
<td>Input and output field name</td>
</tr>
<tr>
<td>k=</td>
<td>Key field name</td>
</tr>
<tr>
<td>s=</td>
<td>Sort field name</td>
</tr>
<tr>
<td>a=</td>
<td>Add item name</td>
</tr>
<tr>
<td>-nfn</td>
<td>CSV without field name</td>
</tr>
<tr>
<td>-nfno</td>
<td>Output without field name</td>
</tr>
<tr>
<td>-x</td>
<td>Specify the field number</td>
</tr>
<tr>
<td>-q</td>
<td>Disable automatic sorting</td>
</tr>
<tr>
<td>precision=</td>
<td>Number of significant figures</td>
</tr>
<tr>
<td>tmpPath=</td>
<td>Work file storage path name</td>
</tr>
<tr>
<td>delim=</td>
<td>Delimiter of vector data</td>
</tr>
<tr>
<td>bufcount=</td>
<td>Number of buffers</td>
</tr>
<tr>
<td>--help</td>
<td>Display help</td>
</tr>
</tbody>
</table>

### 2.9.1 i= Input file name

Specify the name of input file. Most commands only allow a single file to be specified, with the exception of `mcat` command where multiple files can be specified separated with a comma. Yet, certain commands such as `mnewnumber` and `mnewrand` do not require input data.

When this parameter is not defined, data is read from standard input by using pipeline. In the example below, `i=` parameter is not specified for `msum` command because the input data is the result of `msortf`, which is read from standard input through the pipeline.

```bash
$ msortf f=a i=dat.csv | msum k=a f=b o=rsl.csv
```

However, it is difficult to identify errors when results are piped directly from one command to the next. In the following example, `i=` parameter is also specified for `msum`. The results of `msortf` is sent to standard output, and `msum` reads input data from `dat.csv`. Since `msortf` did not add meaning to the input for `msum`, the results from this example is different from the above.

```bash
$ msortf f=a i=dat.csv | msum k=a f=b i=dat.csv o=rsl.csv
```

### Examples

**Example 1: Basic Example**

Run `mcut` using `dat1.csv` as input data.

```bash
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
$ mcut f=customer,amount i=dat1.csv o=rsl1.csv
#ERROR# field name not found: 'customer' in dat1.csv (kgcut)
$ more rsl1.csv
```
Example 2: Specify output field name

Read standard input using redirection ("<").

```
$ mcut f= customer, amount o=rsl2.csv <dat1.csv
#ERROR# invalid argument: customer, (kgcut)
$ more rsl2.csv
rsl2.csv: No such file or directory
```

Related commands

The parameter can be used in all commands except for commands such as mnewnumber and mnewrand.

2.9.2 o= Output file name

Specify the name of output file. Most commands only allow specification of a single file name, with the exception of mtee command where multiple files can be specified. There is also the command that does not require output data, for example, msep.

When this parameter is not defined, data is read from standard input through pipeline. In the following example o= is not specified in msortf because the output data is sent to standard output through pipeline.

```
$ msortf f=a i=dat.csv | msum k=a f=b o=rsl.csv
```

The example below is similar to the above. The difference is that o= parameter is specified for the msortf and the result of msortf is saved to tmp.csv. Even though the two commands are connected with pipeline, there is no data stream from standard output to msum, the receiving process could not read data from pipeline and stays idle.

```
$ msortf f=a i=dat.csv o=tmp.csv | msum k=a f=b o=rsl.csv
```

Below is a more complicated example by using mtee to connect the data streams between the two commands.

```
$ msortf f=a i=dat.csv | mtee o=tmp.csv | msum k=a f=b o=rsl.csv
```

The mtee command writes to a standard input file specified at o= and send the data to standard output concurrently. The results of msortf is written to tmp.csv, at the same time, msum receives the data stream through pipeline from mtee. The final result is saved to rsl.csv.

Examples

Example 1: Basic Example

The result of mcut is saved to rsl1.csv as specified in o= parameter.

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
$ mcut f=customer,amount i=dat1.csv o=rsl1.csv
#ERROR# field name not found: 'customer' in dat1.csv (kgcut)
$ more rsl1.csv
```

Example 2: Redirect

Write to standard input using redirection (">`).

```
```
2.9. SPECIFY PARAMETERS

$ mcut f=customer,amount i=dat1.csv >rsl2.csv
#ERROR# field name not found: 'customer' in dat1.csv (kgcut)
$ more rsl2.csv

Related commands

This parameter can be used in all commands except for certain commands such as sep.

2.9.3  f= Input and output field name

Specify the input and output field name for processing. For example, this parameter specifies the "field name to select" in mcut, "field name to aggregate" for mavg, and "field name to merge" for mjoin. In addition, multiple field names can be specified separated by a comma in between such as f=a,b,c.

The output field name for every specified item from the input file can be renamed in MCMD. This can be done by defining the input field name and output field name separated by a colon in between e.g. f=a:A,b:B. The field name in the output remains the same if the output field name is not specified.

Examples

Example 1: Basic Example

Extract fields val1 and val2.

```bash
$ more dat1.csv
id,val1,val2
A,1,2
B,2,3
C,3,4
$ mcut f=val1,val2 i=dat1.csv o=rsl1.csv
#END# kgcut f=val1,val2 i=dat1.csv o=rsl1.csv
$ more rsl1.csv
val1,val2
1,2
2,3
3,4
```

Example 2: Specify name of output field

Aggregate val1,val2, and rename the fields in the output as sum1,sum2 respectively.

```bash
$ msum f=val1:sum1,val2:sum2 i=dat1.csv o=rsl2.csv
#END# kgs rum f=val1:sum1,val2:sum2 i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,sum1,sum2
C,6,9
```

Related commands

mcut, msum, mcat, mjoin, etc.

2.9.4  k= Key field name

Specify the key field name. A key field uniquely identifies individual rows or an entity in the data, it is used as unit of aggregation, or used as common key for joining fields between two files.

For example, in msum command, aggregate computation is carried out for records with the same key (aggregate key break processing). Whereas in mjoin command, the size of key items in the two data files are compared (join key break processing) and joined.
When \texttt{k=} command is specified, the field(s) specified are first sorted in character string ascending order, afterwards, corresponding processing is carried out.

and is considered as the default field for sorting character strings in ascending order (except for \texttt{mhashsum}). Key break process refers to the processing method for every same key field with the same value assuming that the items are sorted beforehand (However, \texttt{mhashsum} command is an exception).

For details on key break process, please refer to Key break processing. Since frequent sorting may decrease the processing performance, understanding the need for key break processing would help reduce the instances for sorting, desirable for optimizing script performance.

**Examples**

**Example 1: Basic Example**

Compute sum on \texttt{val} column by \texttt{id}.

```bash
$ more dat1.csv
id,val
A,1
B,1
B,2
A,2
B,3
$ msum i=dat1.csv k=id f=val o=rsl1.csv
#END# kgsum f=val i=dat1.csv k=id o=rsl1.csv
$ more rsl1.csv
id%0,val
A,3
B,6
```

**Example 2: Join Process**

Use the join key \texttt{ʠidʡ} from \texttt{dat1.csv}, and join the field \texttt{ʠnameʡ} from \texttt{ref1.csv}.

```bash
$ more dat1.csv
id,val
A,1
B,1
B,2
A,2
B,3
$ more ref1.csv
id,name
A,nysol
B,mcmd
$ mjoin k=id i=dat1.csv m=ref1.csv f=name o=rsl4.csv
#END# kgjoin f=name i=dat1.csv k=id m=ref1.csv o=rsl4.csv
$ more rsl4.csv
id%0,val,name
A,1,nysol
A,2,nysol
B,1,mcmd
B,2,mcmd
B,3,mcmd
```

**Related commands**

\texttt{msum}, \texttt{mslide}, \texttt{mjoin}, \texttt{mrjoin}, \texttt{mcommon}, etc.

**2.9.5 \texttt{s=} Sort Field Name**

Specify the field name for sorting (multiple fields can be specified).
The order of records affects the process results for some commands such as \texttt{maccum}. When \texttt{s=} parameter is specified, sorting is carried out on the specified fields before the processing command.

There are four combinations of sorting methods (order), including numeric / string, and ascending / descending order. The sorting methods can be specified by appending \% followed by \texttt{n} or \texttt{r} after the column name. The examples are as follows.

Character string ascending order: \texttt{field (\% not required)}, character string descending order: \texttt{f=field\%r}, numeric ascending order: \texttt{f=field\%n}, numeric descending order: \texttt{f=field\%nr}.

**Example**

**Example 1: Basic Example**

After sorting by \texttt{id}, calculate the cumulative sum on \texttt{val} column.

```bash
$ more dat1.csv
id,val
A,1
B,2
A,2
B,3

$maccum s=id k=id f=val:val_accum i=dat1.csv o=rsl1.csv
#END# kgaccum f=val:val_accum i=dat1.csv k=id o=rsl1.csv s=id

$ more rsl1.csv
id,val,val_accum
A,1,1
A,2,3
B,1,1
B,2,5
B,3,6
```

**Example 2: Specify sort method**

After sorting the \texttt{val} field in descending numerical order, calculate the cumulative sum on \texttt{val} column.

```bash
$ more dat1.csv
id,val
A,1
B,2
A,2
B,3

$maccum s=id,val\%nr k=id f=val:val_accum i=dat1.csv o=rsl1.csv
#END# kgaccum f=val:val_accum i=dat1.csv k=id o=rsl1.csv s=id,val\%nr

$ more rsl1.csv
id,val,val_accum
A,2,2
A,1,3
B,3,3
B,2,6
B,1,6
```

**Corresponding Commands**

\texttt{maccum, mbest, mmavg, mnumber, mslide, etc.}

**2.9.6 \texttt{a=} Add field name**

Add an additional field (column) according to the field name specified. Most commands add the result in 1 field, thus, only 1 field is specified at this parameter. Nevertheless, \texttt{mcombi} returns multiple fields as output, thus multiple field names are specified delimited by comma.
Examples

Example 1: Basic Example

Add a new field as payday.

```bash
$ more dat1.csv
id
A
B
C
$ msetstr v=20070101 a=payday i=dat1.csv o=rsl1.csv
#END# kgsetstr a=payday i=dat1.csv o=rsl1.csv v=20070101
$ more rsl1.csv
id,payday
A,20070101
B,20070101
C,20070101
```

Example 2: Add multiple fields

Enumerate the two combination of each item A, B, C in the column id.

```bash
$ mcombi f=id n=2 a=id1,id2 i=dat1.csv o=rsl2.csv
#END# kgcombi a=id1,id2 f=id i=dat1.csv n=2 o=rsl2.csv
$ more rsl2.csv
id,id1,id2
C,A,B
C,A,C
C,B,C
```

Related command

mcal, mcombi, mrand, msetstr etc.

2.9.7 -nfn CSV without field names (No Field Names)

This option reads input data without field names. When this option is specified, the field number is used instead of the field name to specify the field. The field number begins from the integer 0 and increments by 1 from the left onwards. When -nfn option is specified, the field name will not be included in the output file.

Examples

Example 1: Basic Example

Extract column 0 and 2.

```bash
$ more dat1.csv
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ mcut -nfn f=0,2 i=dat1.csv o=rsl1.csv
#END# kgcut -nfn f=0,2 i=dat1.csv o=rsl1.csv
$ more rsl1.csv
A,10
A,20
B,15
B,10
B,20
```
Related command

This option can be used in all M-Commands except mchkcsv.

2.9.8 -nfno Output with field names (No Field Names for Output)

This option allows users to remove field names from the output data. Unlike --nfn, this option assumes that input data specified at i= and m= includes field names in the first row.

Examples

Example 1: Basic Example

Extract column0 and 2.

```
$ more dat1.csv
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ mcut -nfn f=0,2 i=dat1.csv o=rsl1.csv
#END# kgcut -nfn f=0,2 i=dat1.csv o=rsl1.csv
$ more rsl1.csv
A,10
A,20
B,15
B,10
B,20
```

Related commands

This option can be used in all commands except mchkcsv.

2.9.9 -x Specify by item number

This option allows users to specify a column with corresponding field number where input data includes field names. Users can specify the output field name(s) by adding colon right after input field, followed by the output field name.

Examples

Example 1: Basic Example

Compute the sum of all items in column 1 and 2 of the same key.

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ msum -x k=0 f=1,2 i=dat1.csv o=rsl1.csv
#END# kgsum -x f=1,2 i=dat1.csv k=0 o=rsl1.csv
$ more rsl1.csv
customer,quantity,amount
A,3,30
B,5,45
```
Example 2: Output column names

Rename column 1 and 2 as a, b respectively.

```bash
$ mvsum -x k=0 f=1:a,2:b i=dat1.csv o=rsl2.csv
#END# kgsum -x f=1:a,2:b i=dat1.csv k=0 o=rsl2.csv
$ more rsl2.csv
customer,a,b
A,3,30
B,5,45
```

Example 3: Error when using -nfn

The `-nfn` option assumes data starts from the first row when computing the sum of "quantity" and "amount". However, the result will not be computed as expected since the position of first row of data is defined differently when using `-x` and `-nfn`.

```bash
$ mvsum -nfn k=0 f=1,2 i=dat1.csv o=rsl3.csv
#END# kgsum -nfn f=1,2 i=dat1.csv k=0 o=rsl3.csv
$ more rsl3.csv
customer,0,0
A,3,30
B,5,45
```

Related commands

This option can be used in all commands except mchkcsv.

2.9.10 -q Disable Automatic Sorting

Use this option to disable automatic sorting on fields specified at `k=` parameter.

The `s=` option is not required when `k=` parameter is defined at the same time, therefore, each command operates the same as MCOMMAND Ver. 1.0.

Example

Example 1: Basic Example

Find out the cumulative value by `id` field. When `-q` option is specified, sorting by field specified at `k=` parameter will be disabled.

```bash
$ more dat1.csv
id,val
A,1
B,1
B,2
A,2
B,3
$ maccum -q k=id f=val:val_accum i=dat1.csv o=rsl1.csv
#END# kgaccum -q f=val:val_accum i=dat1.csv k=id o=rsl1.csv
$ more rsl1.csv
id,val,val_accum
A,1,1
B,1,1
B,2,3
A,2,2
B,3,3
```

Corresponding Commands

This function is available in all commands where `k=` parameter exists.
2.9. SPECIFY PARAMETERS

2.9.11 precision= Number of significant digits

Applies sprintf format ["%.ng"] in C language. This format converts the number of significant figures defined from normalized notation (integer bits, decimal bits: ex. 123.456) to exponent notation (mantissa e± exponent part: ex. 1.23456e+02). The criteria to adopt exponent notation for conversion is when the exponent bits exceed the specified number of significant digits or if it is less than or equal to -5 (i.e more than 4 zeros after decimal points).

Integers between 1 to 16 can be specified in n, the default value is 10. When n < 1, set n = 1, and when n > 16 set to n = 16.

In addition, the number of significant figures can be changed by setting the environment variable KG_Precision. However, changes to the environment variable will affect the execution of all commands.

Examples

Example 1: Basic Example

The exponential notation of id=1 is 1.2345678e +08, the exponent bits is more than 6 significant figures when the significant figures of mantissa is set at 6. The exponential notation of id=2 is 1.23456789e +03, the exponent bits is more than 7 significant figures when the significant figures of integer bits + decimal bits is set at 6. The exponential notation of id=4 is 1.23456789e-04, the exponent bits is less than -4 when the significant figures is set at 6. The exponential notation of id=5 is 1.23456789e-05, the exponent bits is less than -4 when the significant figures of mantissa is set at 6.

Example 2: Case when precision=2

Example 3: Specify the environment variable

When the environment variable is set, the setting will be applied to all commands in subsequent processes.
Related commands

This setting applies to all commands for calculating real numbers which is used in msum, mcal.

### 2.9.12 tmpPath= Path name of temporary file

Specify the name of the directory which stores the temporary files for use by the command. For example, the results from msortf is saved as a temporary file during partitioned sort. If the path is not specified, the file is saved in /tmp. The name of temporary files begins with __KG_TMP.

The temporary files are deleted if the command terminates normally (includes termination by exit signal, or termination by signal from MCMD signal). Temporary files will be retained in the directory when the program is terminated unexpectedly by power outage or bug.

Depending on the amount of data, enormous amount of temporary data may be generated (more than 1 million files). This will significantly slow down the execution of commands, therefore, clean out the files in the temporary path on a regular basis. Currently there is no plans to implement functions for garbage collection to remove objects no longer used by the program.

The temporary directory can be changed by setting the environment variable KG_Tmp_Path, however, the same variable applies to the execution of all commands.

**Examples**

**Example 1: Basic Example**

Set the tmp directory under the current directory for temporary files.

```
$ msortf f=val tmpPath=./tmp i=dat1.csv o=rsl1.csv
#END# kgsortf f=val i=dat1.csv o=rsl1.csv tmpPath=./tmp
```

**Example 2: Specify the environment variable**

The settings of the environment variable will be applied to subsequent commands.

```
$ export KG_TmpPath="~/tmp
$ msortf f=val i=dat1.csv o=rsl1.csv
#ERROR# internal error: cannot create temp file (kgsortf)
```

Related commands

This applies to commands such as msortf and mdelnull which select records by key field, and commands such as mbucket, mnjoin, and mshare that require multiple pass scanning based on key field.

### 2.9.13 delim= Delimiter of vector element

Specify the delimiter for elements in vector data. The default delimiter is 1 byte space. When comma is specified as the delimiter for the vector, the vector is enclosed in double quotes to avoid confusion with the comma delimiter in CSV file.
Examples

Example 1: Basic Example

Sort the elements of the vector field \( \text{vec} \) with colon as a delimiter.

```bash
$ more dat1.csv
vec
b:a:c
x:p
$ mvsort vf=vec delim=: i=dat1.csv o=rsl1.csv
#END# kgvsort delim=: i=dat1.csv o=rsl1.csv vf=vec
$ more rsl1.csv
vec
a:b:c
p:x
```

Example 2: When delim parameter is not specified

Since delim parameter is not specified, \( b:a:c \) and \( x:p \) is interpreted as one element.

```bash
$ mvsort vf=vec i=dat1.csv o=rsl2.csv
#END# kgvsort i=dat1.csv o=rsl2.csv vf=vec
$ more rsl2.csv
vec
b:a:c
x:p
```

Example 3: Use comma as delimiter

If comma is used as delimiter for the vector, the entire vector is enclosed by double quote to draw distinction between the delimiter of CSV and the delimiter of the vector.

```bash
$ more dat2.csv
id,vec1,vec2
1,a,b
2,p,q
$ mvcat vf=vec1,vec2 a=vec3 delim=, i=dat2.csv o=rsl3.csv
#END# kgvcat a=vec3 delim=, i=dat2.csv o=rsl3.csv vf=vec1,vec2
$ more rsl3.csv
id,vec3
1,"a,b"
2,"p,q"
```

Related commands

This parameter can be used in all vector related commands such as such as `mvcat` and `mvsort`.

2.9.14 bufcount= Buffer size

Specify the internal buffer size (number of blocks) to be used in commands such as mbucket, mnjoin, and mshare, for processing key units at which data requires multiple pass scanning. One buffer block contains 4MB, the default size is 10 blocks (40MB). In case of buffer overflow, data is written to a temporary file. If the key size is very large, the processing speed can be improved by adjusting this parameter if memory permits.

Examples

Example 1: Basic Example

If the key size of the reference file is less than 80MB (4MB \( \Diamond 20 \)), the temporary file will not be used.
$ mnjoin k=id m=ref.csv f=name i=dat.csv o=rsl.csv bufcount=20 
#END# kgnjoin bufcount=20 f=name i=dat.csv k=id m=ref.csv o=rsl.csv

Related command

Commands that require multiple pass scanning of the data to process key units, such as mbucket, mnjoin, and mshare.
2.10 Environment Variable

Shell environment variable can be set in MCMD to customize the settings for commands. The environment variables that can be set for MCMD are listed in Table 2.5.

Table 2.5: List of the environment variable to configure KGMOD

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KG_iSize</td>
<td>4096000</td>
<td>Size of data read for each time. Four times of KG_iSize is allocated for input buffer. However, 10 times of the command parameters is allocated for kgsortf. Refer to KG_BlockCount for command lks buffer size on processing by key block. The following conditions must be satisfied. KG_iSize = KG_MaxRecLen i (given i is an integer greater than 1) KG_iSize &gt; KG_MaxRecLen + 2</td>
</tr>
<tr>
<td>KG_oSize</td>
<td>2048000</td>
<td>Size of data written for each time. The same memory size in KG_oSize is allocated for output buffer. KG_oSize = KG_MaxRecLen i (given i is an integer greater than 1) KG_oSize &gt; KG_MaxRecLen + 2</td>
</tr>
<tr>
<td>KG_MaxRecLen</td>
<td>1024000</td>
<td>The maximum number of characters per row (maximum:1024000) It must satisfy the conditions shown in KG_iSize and KG_oSize.</td>
</tr>
<tr>
<td>KG_BlockCount</td>
<td>128</td>
<td>The number of buffers required when processing by key. The column KG_iSize shows buffer size*KG_BlockCount is allocated. KG_MaxRecLen = 2 * KG_iSize + 4 * KG_oSize * KG_BlockCount</td>
</tr>
<tr>
<td>KG_TempPath</td>
<td>/tmp</td>
<td>The directory of temporary files in the default directory used by library functions.</td>
</tr>
<tr>
<td>KG_Precision</td>
<td>10</td>
<td>Significant figures</td>
</tr>
<tr>
<td>KG_VerboseLevel</td>
<td>4</td>
<td>Output error message for M-Command 0: Do not print any messages. 1: + error message output 2: + warning message output 3: + end message output 4: + msg message output (default)</td>
</tr>
</tbody>
</table>

Note 1) Processing by key, such as mnjoin, refers to reading data with same key into the memory at once. Refer to the manual of corresponding command for more details.

Below is an example of changing the settings of KG_VerboseLevel to control the command message.

Examples

Example 1: End message

The default setting of output message is set as KG_Verbose=4 which displays a message when the program finished processing normally or terminates upon an error.

```
$ more dat.csv
k,v
A,1
B,2
$ mcut f=k,v i=dat.csv o=out.csv
#END# kgcut f=k,v i=dat.csv o=out.csv
$ mcut x=k,v i=dat.csv o=out.csv
#ERROR# unknown parameter x= (kgcut)
```

Example 2: Only display error messages

If KG_Verbose=1, the stop error message is displayed, but not the normal exit message.

```
$ export KG_VerboseLevel=1
$ mcut f=k,v i=dat.csv o=out.csv
$ mcut x=k,v i=dat.csv o=out.csv
#ERROR# unknown parameter x= (kgcut)
```
Example 3: Do not display any messages

If \texttt{KG\_Verbose}=0, no message will be displayed.

\begin{verbatim}
$ export KG_VerboseLevel=0
$ mcut f=k,v i=dat.csv o=out.csv
$ mcut x=k,v i=dat.csv o=out.csv
\end{verbatim}
2.11 Key Break Processing

In key break processing, it is assumed that within the column which matches the column specified, processing is executed for key fields with the same value. Key break processing is broadly divided into two types of processes. First is key break processing for aggregate calculation (referred to as “aggregate key break processing” below), second is key break processing for joins (referred to as “join key break processing” below).

Join key break processing is executed for commands such as `mjoin`, `mcommon` which contains the word “join” and “common”. Aggregate key break processing is carried out on other commands with `k=` parameter.

For example, when `msum` command triggers aggregate key break processing, it detects the change of value in the key field, and executes aggregate processing for records with the same key. Therefore, it is necessary to sort the records by key field beforehand (unless input file is sorted in advance), and sorting is carried out in `msum` command before aggregate processing.

Join key break processing involves a more complicated process. For instance, `mjoin` command takes in two data files, and compares the values in the key field. The key fields from the smaller data set is read continuously, and the records are joined when the key fields in input file and reference file matches. When the comparison of key fields values, since key break processing is used for join operation, the key fields from the two data files need to be sorted beforehand. Therefore, in this version, the two data files used in `mjoin` commands are sorted.

Basic sorting character string ascending order is carried out for both key break processing, however, when joining records by numerical range in `mrjoin` command, sorting is carried out by numeric ascending order.

Besides the fields defined at `k=` parameter are automatically sorted, in other commands automatic sorting is pre-determined, thus users do not need to resolve whether the input files requires sorting. Even though users no longer need to initiate the sort command, note that sorting is handled within each command internally. Thus, depending on the construction of the script, sort processing may frequently take place which could reduce performance.

Example of Script

Example of script when sorting takes place frequently

Initially, `name` column is sorted and saved as `xxcustomer` output file, afterwards, join processing by `id` key field is carried out by `mjoin` command. In this case, `mjoin` is executed three times, and `id` column from `xxcustomer` inputer data is sorted at each instance of `mjoin` command.

```
| mcut  i=customer.csv f=id,name  |
| msort f=name o=xxcustomer        |
| mjoin i=xxcustomer m=address.csv k=id f=address o=cust_address.csv |
| mjoin i=xxcustomer m=phone.csv k=id f=phone o=cust_phone.csv        |
| mjoin i=xxcustomer m=age.csv k=id f=age   o=cust_age.csv            |
```

Example of script to minimize sorting

When the script is modified as follows, since `xxcustomer` file is sorted by `id` field and saved as `xxcustomer`. Automatic sorting of the input file at `mjoin` commands is not carried out.

```
| mcut  i=customer.csv f=id,name  |
| msort f=id o=xxcustomer         |
| mjoin i=xxcustomer m=address.csv k=id f=address o=cust_address.csv |
| mjoin i=xxcustomer m=phone.csv k=id f=phone o=cust_phone.csv        |
| mjoin i=xxcustomer m=age.csv k=id f=age   o=cust_age.csv            |
```
Chapter 3

Command Reference
3.1 Format of MCMD

The format for all commands is explained in this section. The format of all command references is shown as in the following example.

Format

```
```

Parameters

- **k=:** List of field name(s) to match with the input data
  - [join key break processing: character string in ascending order].
  - The field(s) in the input data is specified at the K= parameter.
  - Field in the reference data will be combined with same field in the record.
  - If the field(s) specified at K= from reference file do not match any values, it is treated as a NULL value.
- **f=:** Specify the list of field name(s) to join from the reference file.
  - When the f= parameter is not defined, all fields except the key field(s) in the reference file will be joined to the input file.

Command options and parameter can be specified after the command name. The parameter takes a value and the parameter name and value is separated by an `=` sign. An option begins with one dash `-` or two dash `--`. Command options and parameters that are common in most commands include `-nfn`. The link to that section is appended in a separate section. The description of each parameter is described in the format below.

Parameters enclosed in square brackets like `[f=]` means that it is optional. On the other hand, parameters that are not enclosed in square brackets like `k=` means that it is a required parameter. Parameters separated by a vertical bar enclosed in square parentheses, such as `[to=|size=]` means that the command will only read either `to=` or `size=` (e.g. `mbest` command). On the other hand, if the parameters separated by `|` is not enclosed in square brackets such as `m=|i=`, one of these parameters is required and must be specified (e.g. `mjoin` command. See above).

Further, parameters or options are only required when specifying certain options, there are also more complex parameter conditions which is described in the description field of each parameter.
3.2 maccum Cumulative Calculation

Calculates the cumulative value for the column specified at f= parameter and save the result in a new column. Cumulative calculation is carried by the key unit when k= is specified.

Format

maccum f= [s=] [k=] [i=] [o=] [-nfn] [-nfno] [-x] [-q] [precision=] [--help] [--version]

Parameter

f= Specify the field(s) (multiple fields can be specified) for cumulative calculation. Field with NULL values is ignored.

s= After sorted by specified field (multiple fields can be specified), calculate cumulative value. s= parameter is required if -q option is not specified.

k= Specify the field name as unit for cumulative calculation (multiple fields can be specified).

Example

Example 1: Basic Example

Calculate the cumulative values of "Quantity" and "Amount" fields for each "Customer", save output as new data attributes in new columns named "AccumQuantity" and "AccumAmount".

```bash
$ more dat1.csv
Customer,Quantity,Amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ maccum s=Customer f=Quantity:AccumQuantity,Amount:AccumAmount i=dat1.csv o=rsl1.csv
#END# kgaccum f=Quantity:AccumQuantity,Amount:AccumAmount i=dat1.csv o=rsl1.csv s=Customer
$ more rsl1.csv
Customer,Quantity,Amount,AccumQuantity,AccumAmount
A,1,10,1,10
A,2,20,3,30
B,1,15,4,45
B,3,10,7,55
B,1,20,8,75
```

Example 2: Specify Calculation by Key

Calculates the cumulative value of "Quantity" and "Amount" fields for each "Customer", and save the output in new columns named "AccumQuantity" and "AccumAmount".

```bash
$ more dat1.csv
Customer,Quantity,Amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ maccum k=Customer s=Customer f=Quantity:AccumQuantity,Amount:AccumAmount i=dat1.csv o=rsl1.csv
#END# kgaccum f=Quantity:AccumQuantity,Amount:AccumAmount i=dat1.csv k=Customer o=rsl1.csv s=Customer
$ more rsl1.csv
Customer,Quantity,Amount,AccumQuantity,AccumAmount
A,1,10,1,10
A,2,20,3,30
B,1,15,1,15
```
Example 3: Cumulative computation with NULL values

Calculate the cumulative values of "Quantity" and "Amount" item, and save the output as new columns named "AccumQuantity" and "AccumAmount". NULL values are ignored. Records with NULL values will be retained in the output.

```
$ more dat2.csv
Customer,Quantity,Amount
A,1,10
A,,20
B,1,15
B,3,
B,1,20
$ maccum s=Customer f=Quantity:AccumQuantity,Amount:AccumAmount i=dat2.csv o=rsl3.csv
#END# kgaccum f=Quantity:AccumQuantity,Amount:AccumAmount i=dat2.csv o=rsl3.csv s=Customer
$ more rsl3.csv
Customer%0,Quantity,Amount,AccumQuantity,AccumAmount
A,1,10,1,10
A,,20,,30
B,1,15,2,45
B,3,,5,
B,1,20,6,65
```
### 3.3 marff2csv - Conversion from arff to csv Format

Convert data from arff format (data format for WEKA) to csv format.

#### arff format data

The arff data format is described below.

```plaintext
@RELATION Title
@ATTRIBUTE Field name string(character string)
@ATTRIBUTE Field name date(date format: format is optional.
If not defined, "yyyy-MM-dd'T'HH:mm:ss"
)
@ATTRIBUTE Quantity numeric(number)
@ATTRIBUTE Product {A,B}(category field type)

@DATA(Actual data)
No.1,20081201,1,10,A
No.2,20081202,2,20,A
No.3,20081203,3,30,A
No.4,20081201,4,40,B
No.5,20081203,5,50,B
```

#### Format

```
marff2csv [i=] [o=} [-fn] [-nfno] [--help] [--version]
```

#### Example

**Example 1: Basic Example**

Convert customer purchasing data in arff format to csv format.

```plaintext
$ more dat1.arff
@RELATION Customer Purchase Data
@ATTRIBUTE Customer string
@ATTRIBUTE Date date yyyyMMdd
@ATTRIBUTE Quantity numeric
@ATTRIBUTE Amount numeric
@ATTRIBUTE Product {A,B}

@DATA
No.1,20081201,1,10,A
No.2,20081202,2,20,A
No.3,20081203,3,30,A
No.4,20081201,4,40,B
No.5,20081203,5,50,B
$ marff2csv i=dat1.arff o=rsl1.csv
```

#### Related Command

```
mcsv2arff
```
Reference

http://weka.wikispaces.com/ARFF
3.4 MAVG - CALCULATE AVERAGE

Calculates the average values in column specified by the f= parameter.

Format

mavg f= [k=] [-n] [i=} [o=] [-nfn] [-nfn] [-x] [-q] [precision=] [--help] [--version]

Parameters

f= Specify the field(s) (multiple fields can be specified) with the values to be aggregated.
    Use : (colon) to specify the new field name. Example: f=Quantity:AverageQuantity
k= Specify the set of field name(s) (Multiple fields can be specified) as unit of aggregation.
   -n Output as NULL if the data consist at least one NULL value.

Examples

Example 1: Basic Example

Calculate the average values of "Quantity" and "Amount" fields for each "Customer", save the computed output in new columns named "AverageVolume" and "AverageAmount".

```bash
$ more dat1.csv
Customer,Quantity,Amount
A,1,5
A,2,20
B,1,15
B,,10
B,5,20
$ mavg k=Customer f=Quantity:AvgQuantity,Amount:AvgAmount i=dat1.csv o=rsl1.csv
#END# kgavg f=Quantity:AvgQuantity,Amount:AvgAmount i=dat1.csv k=Customer o=rsl1.csv
$ more rsl1.csv
Customer%0,AvgQuantity,AvgAmount
A,1.5,12.5
B,3,15
```

Example 2: Output consisting of NULL values

Calculate the average values of "Quantity" and "Amount" fields for each "Customer", save output in a new columns named "AverageVolume" and "AverageAmount". When specifying the -n option, if a NULL value is included in the input, the result will return NULL value.

```bash
$ mavg k=Customer f=Quantity:AvgQuantity,Amount:AvgAmount -n i=dat1.csv o=rsl2.csv
#END# kgavg f=Quantity:AvgQuantity,Amount:AvgAmount -n i=dat1.csv k=Customer o=rsl2.csv
$ more rsl2.csv
Customer%0,AvgQuantity,AvgAmount
A,1.5,12.5
B,,15
```

Example 3: Calculate sum without key field

Calculate the average values of "Quantity" and "Amount" fields, and save the outputs in columns "AvgQuantity" and "AvgAmount".

```bash
$ mavg f=Quantity:AvgQuantity,Amount:AvgAmount i=dat1.csv o=rsl3.csv
#END# kgavg f=Quantity:AvgQuantity,Amount:AvgAmount i=dat1.csv o=rsl3.csv
$ more rsl3.csv
Customer,AvgQuantity,AvgAmount
A,1.5,12.5
B,15
```


Related Commands

mhashavg : Aggregate calculation does not require prior sorting on the key field.
msum : Command to calculate sum.
mstats : Calculate a variety of statistics.
3.5 mbest - Select Rows

Select records based on the specified row numbers. Note that row number starts at 0 (the first row of data starts at row 0 excluding the row of field names). Define the row numbers at from= and to= parameters (and size= parameter in some instances).

Format

```plaintext
mbest [s=] [from=] [to=|size=] [k=] [u=] [-r] [i=] [o=] [-nfn] [-nfno] [-x] [-q] [--help] [--version]
```

Parameters

- **s=**  After sorted by specified field(s) (multiple fields can be specified), the rows are selected. 
  - s= parameter is required if -q option is not specified.
- **from=**  Define the start of row number (integers greater than 0) [default value: 0]
- **to=**  Define the end of row number (integers greater than 0) [default value: 0]
  - Where [value of from=] ≤ [value of to=].
- **size=**  Number of rows to select [default value: 1]
  - to= and size= cannot be specified at the same time.
- **k=**  Key field (accept multiple key fields) [aggregate key break processing]
  Records with same key values will be selected based on the defined rows at from=,to=,size=.
  - -x,-nfn options can be used to specify the field number (0~).
- **u=**  Output file of unmatched records.
  - Unmatched records that do not match the criteria is written to the defined file.
- **-r**  Reverse selection
  - Select the rows other than the ones defined at the parameter from=,to=(size=).

Example

Example 1: Basic Example

This example assumed that the "quantity" and "amount" fields are sorted from the largest value (descending order). Records are selected from the first row (line 0) by default if from=,to=,size= parameters are not specified.

```plaintext
$ more dat1.csv
Customer,Quantity,Amount
A,20,5200
B,18,4000
C,15,3500
D,10,2000
E,3,800
$ mbest s=Quantity%nr,Amount%nr i=dat1.csv o=rsl1.csv
#END# kgbest i=dat1.csv o=rsl1.csv s=Quantity%nr,Amount%nr
$ more rsl1.csv
Customer%0nr,Quantity%0nr,Amount%0nr
A,20,5200
```

Example 2: Basic Example 2

After sorting by "customers", select 3 rows from the first row (line 0).

```plaintext
$ mbest s=Customer from=0 size=3 i=dat1.csv o=rsl2.csv
#END# kgbest from=0 i=dat1.csv o=rsl2.csv s=Customer size=3
$ more rsl2.csv
Customer%0,Quantity,Amount
A,20,5200
B,18,4000
C,15,3500
```
CHAPTER 3. COMMAND REFERENCE

Example 3: Basic Example 3

Without sorting (in the original order), select from line 0 to line 1.

```
$ mbest -q from=0 to=1 i=dat1.csv o=rsl3.csv
#END# kgbest -q from=0 i=dat1.csv o=rsl3.csv to=1
$ more rsl3.csv
Customer,Quantity,Amount
A,20,5200
B,18,4000
```

Example 4: Reverse Selection

Select records other than customers’ first visit to store. Save the records of customers’ first visit to the file fvd.csv.

```
$ more dat2.csv
Customer,Date,Amount
A,20081201,10
A,20081207,20
A,20081213,30
B,20081002,40
B,20081209,50
$ mbest s=Customer,Date k=Customer -r u=fvd.csv i=dat2.csv o=rsl4.csv
#END# kgbest -r i=dat2.csv k=Customer o=rsl4.csv s=Customer,Date u=fvd.csv
$ more rsl4.csv
Customer,Date,Amount
A,20081207,20
A,20081213,30
B,20081209,50
$ more fvd.csv
Customer,Date,Amount
A,20081201,10
B,20081002,40
```

Related commands

- **msel**: The line() function can be specified in the condition parameter to carry out similar processing functions.
- **muniq**: Returns unique values in key field.
- **mselnum**: Select rows within a numeric range.
3.6 mbucket - Partition Data into Uniform Buckets

Partition numerical data field(s) specified at \( f = \) into a number of segments specified by \( n = \). There are two ways to compute the bucket intervals. The first method is to compute an uniform spread of data points for each partition (referred to as partition of uniform buckets). The second method is to compute uniform interval ranges across partitions (referred to as partition of uniform ranges). Data is partitioned into equal interval ranges when the \(-\text{rng}\) option is specified. Otherwise, data will be partitioned uniformly across buckets if this option is not specified. When multiple fields are defined at \( f = \), the data buckets are generated separately for each field.

Format

\[
\text{mbucket } f = n = [-\text{rng}] [-r] [F=] [k=] [O=] [i=] [o=] [bufcount=] [-nfn] [-nfno] [-x] [-q] [precision=] [--help] [--version]
\]

Parameters

- \( f = \) Partitioning is based on the value specified in this field (multiple fields can be specified).
- \( n = \) Number of buckets
  - Specify the number of buckets to be partitioned for the field(s) defined in \( f = \) parameter(s).
  - If 1 is defined, the command will partition by the number of items specified in \( f = \) parameter.
- \( F = \) Output format [default value: 0]
  - Output format of bucket label.
    - 0: Display bucket numbers
    - 1: Display value range of buckets
    - 2: Display both bucket numbers and value range of buckets.
- \( k = \) Key field(s) to retrieve rows of data incrementally for bucket partitions (multiple keys can be specified).
- \( O = \) Output file with values range of bucket
  - Specify output file name with values range of bucket on the field name(s) defined in parameter \( f = \).
- \(-\text{rng}\) Define equal value of bucket range
  - Divide buckets by the specified value range.
- \(-r\) Print bucket numbers in reverse order.

Examples

Example 1: Basic Example

Partition \( x \) and \( y \) into two subsets of equal extent and save the output file as \( \text{rng.csv} \)

\[
\begin{align*}
\text{Example 1: Basic Example} \\
\end{align*}
\]

\[
\begin{align*}
\text{Partition } x \text{ and } y \text{ into two subsets of equal extent and save the output file as } \text{rng.csv} \\
\end{align*}
\]

\[
\begin{align*}
\text{Examples} \\
\end{align*}
\]

\[
\begin{align*}
\text{Example 1: Basic Example} \\
\end{align*}
\]

\[
\begin{align*}
\text{Partition } x \text{ and } y \text{ into two subsets of equal extent and save the output file as } \text{rng.csv} \\
\end{align*}
\]

\[
\begin{align*}
\text{Examples} \\
\end{align*}
\]

\[
\begin{align*}
\text{Example 1: Basic Example} \\
\end{align*}
\]
Example 2: Partition by equal range

Use the `-rng` option to partition the data by uniform value ranges.

```bash
$ mbucket f=x:b,y:y n=2 -rng O=rng2.csv i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,x,y,xb,yb
A,2,7,1,2
B,6,7,2,2
C,5,6,2,2
D,7,5,2,2
E,6,4,2,2
F,1,3,1,1
G,3,3,1,1
H,4,2,2,1
I,7,2,2,1
J,2,1,1,1
$ more rng2.csv
fieldName,bucketNo,rangeFrom,rangeTo
x,1,,4
x,2,4,
y,1,,3
y,2,3,
```

Example 3: Example using key field

Partition `x` and `y` into two subsets of equal extent using "id" as the key parameter. By specifying `n=2,3`, field `x` is divided into 2 buckets, and field `y` is divided into 3 buckets. Include bucket numbers and value range of buckets in the output file (`F=2`).

```bash
$ more dat2.csv
id,x,y
A,2,7
A,6,7
A,5,6
B,7,5
B,6,4
B,1,3
C,3,3
C,4,2
C,7,2
C,2,1
$ mbucket k=id f=x:y:n=2,3 F=2 i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id%0,x,y,xb,yb
A,2,7,1:3.5,2:6.5
A,6,7,2:3.5,1:6.5
A,5,6,3.5,1:6.5
B,7,5,3:4.5
B,6,4,2:3.5
B,1,3,1:3.5
C,3,3,1:3.5,3.5
C,4,2,2:3.5,2:1.5
C,7,2,2:3.5,2:1.5
C,2,1,1:3.5,1:1.5
```
3.6. MBUCKET - PARTITION DATA INTO UNIFORM BUCKETS

Theorem of Arithmetic

Assume $D$ is a data set of $n$ elements $(x_1, x_2, \ldots, x_n)$. Partition $D$ uniformly into $k$ number of groups (referred to as buckets) and ensure that the data is partitioned uniformly in each bucket. Dispersion is used as the fundamental evaluation criteria for uniformity.

Let $X = \{x_i \mid 1 \leq i \leq n, \ x_i \in D\}$

Arrange each value within $X$ in ascending order $v_1, v_2, \ldots, v_n$. Partition of the buckets $X$ is represented by partition of intervals $\{v_1, v_2, \ldots, v_n\}$. This interval is defined as $I_1, I_2, \ldots, I_k$. Based on these elements, $D_j$, $n_j$, the following is defined.

$$D_j = \{x_i \mid 1 \leq i \leq n, \ x_i \in I_j\}$$

$$n_j = |D_j|$$

The basis of uniform dispersion is described as follows.

$$Var = \sum_{j=1}^{k} (n_j - \bar{n})$$

If $\bar{n} = n/k$, the equation becomes

$$Var = \sum_{j=1}^{k} (n_j^2 - 2n_j\bar{n} + \bar{n}^2) = \sum_{j=1}^{k} n_j^2 - kn^2$$

$k\bar{n}^2$ remains a constant regardless of how the data is partitioned. Therefore, the first term from the above equation:

$$Var' = \sum_{j=1}^{k} n_j^2$$

minimizes $Var$ and splits the data into intervals.

Algorithm

Recursive equation for dynamic programming is used for optimization as illustrated in the following. The minimum value of $\sum_{j=1}^{h} n_j^2$ is divided into $h$ intervals of $I_1, I_2, \ldots, I_h$ of $v_1, v_2, \ldots, v_m$, to determine $DP(n, k)$. $DP(m, h)$ is substituted in the following function.

$$DP(m, h) = \min_{g = h-1, \ldots, m-1} \{ DP(g, h-1) + |\{ x_i \mid v_{g+1} \leq x_i \leq v_m \}|^2 \}$$

The above function recursively defines a sequence. Given the initial term below:

$$DP(m, 1) = |\{ x_i \mid v_1 \leq x_i \leq v_m \}|^2, \ m = 1, \ldots, n$$

the next term of the sequence is defined as a function of the preceding term and iterated as follows: $DP(m, 2)(m = 1, \ldots, n), DP(m, 3)(m = 1, \ldots, n), \ldots, DP(m, k-1)(m = 1, \ldots, n)$.

The function ends until it searches for the term $DP(n, k)$

$$DP(m, k) = \min_{g = k-1, \ldots, n-1} \{ DP(g, k-1) + |\{ x_i \mid v_{g+1} \leq x_i \leq v_n \}|^2 \}$$

Related command

mmbucket : Partition multidimensional data into uniform buckets
CHAPTER 3. COMMAND REFERENCE

3.7 mcat - ConCATenate

Merge all records in the files specified at \texttt{i=} parameter according to the order of files. If a wild card is used to specify file names, the files will be merged in alphabetical order of the file name.

Format

\texttt{mcat \{f=\} \{-skip_fnf\} \{-nostop\|-skip\|-force\} \{i=\} \{o=\} \{-nfn\} \{-nfno\} \{-x\} \{---help\} \{---version\}}

Parameters

\begin{itemize}
  \item \texttt{i=} Specify list of input file names.
  \item \texttt{f=} Specify the field name(s) to concatenate.
  \item \texttt{-skip_fnf} If a specified file in the \texttt{i=} parameter does not exist, the program will bypass the error. However, the program returns an error if all files cannot be found.
  \item \texttt{-nostop} \texttt{-nostop,\-skip,\-force} are parameters for controlling exceptions when header is not present. \texttt{-nostop} flag returns null if field name is not specified. When \texttt{-nfn} flag is used with \texttt{stop} flag, the program terminates if the number of items in the data is different than the parameter defined.
  \item \texttt{-skip} Files are not concatenated if field name(s) is not specified. When \texttt{-nfn} flag is used with \texttt{-skip} flag, files are not concatenated if the number of data items are different.
  \item \texttt{-force} Force concatenation of files using location of fields when header is not present. Print output to null if item number is not available.
  \item \texttt{-stdin} Merge from standard input.
  \item \texttt{-add_fname} Add file name in the last column.
\end{itemize}

Note

\begin{itemize}
  \item Wild card characters (“?” and “*”) can be used to specify multiple directory and file names.
  \item The symbol “/” can be used to indicate home directory.
  \item The files are concatenated according according to the order specified in the \texttt{i=} parameter. If a wild card is used, files will be merged in alphabetical order. Standard input takes precedence when merging files.
\end{itemize}

Examples

Example 1: Concatenate files with the same header

\begin{verbatim}
$ more dat1.csv
customer,date,amount
A,20081201,10
B,20081002,40
$ more dat2.csv
customer,date,amount
A,20081207,20
A,20081213,50
B,20081209,50
$ mcat i=dat1.csv,dat2.csv o=rsl1.csv
#END# kgcat i=dat1.csv,dat2.csv o=rsl1.csv
$ more rsl1.csv
customer,date,amount
A,20081201,10
B,20081002,40
A,20081207,20
\end{verbatim}
Example 2: Concatenate files with different header

The first file `dat1.csv` defined at `i=` contains columns "customer,date,amount". However, since "amount" is not present in `dat3.csv`, it will return an error. Nevertheless, the contents in the first file `dat1.csv` is merged and saved in the output.

```sh
$ more dat3.csv
customer,date,quantity
A,20081201,3
B,20081002,1
$ mcat i=dat1.csv,dat3.csv o=rsl2.csv
#ERROR# field name [amount] not found on file [dat3.csv] (kgcat)
$ more rsl2.csv
customer,date,amount
A,20081201,10
B,20081002,40
```

Example 3: Concatenate files with different header

When previous example is attached with `-nostop` option, the command will continue processing and return NULL value for the data item not found. Other options such as `skip,force` handle conditions when the field name is not found. For details, refer to the description of parameters.

```sh
$ more dat3.csv
customer,date,quantity
A,20081201,3
B,20081002,1
$ mcat -nostop i=dat1.csv,dat3.csv o=rsl3.csv
#END# kgcat -nostop i=dat1.csv,dat3.csv o=rsl3.csv
$ more rsl3.csv
customer,date,amount
A,20081201,10
B,20081002,40
A,20081201,10
B,20081002,
```

Example 4: Concatenate specific field names from input files

Merge field names specified at `f=`.

```sh
$ mcat f=customer,date i=dat2.csv,dat3.csv o=rsl4.csv
#END# kgcat f=customer,date i=dat2.csv,dat3.csv o=rsl4.csv
$ more rsl4.csv
customer,date
A,20081207
A,20081213
B,20081209
A,20081201
B,20081002
```

Example 5: Merge from standard input

Read file `dat2.csv` from standard input by specifying `-stdin` option.

```sh
$ mcat -stdin i=dat1.csv o=rsl5.csv <dat2.csv
#END# kgcat -stdin i=dat1.csv o=rsl5.csv
$ more rsl5.csv
customer,date,amount
A,20081207,20
A,20081213,30
B,20081209,50
A,20081201,10
B,20081002,40
```
Example 6: Add file name as new column

When `-add_fname` is specified, the original file name `fileName` is added as a new column. File name of standard input is `/dev/stdin`.

```
$ mcat -add_fname -stdin i=dat1.csv o=rsl6.csv <dat2.csv
#END# kgcat -add_fname -stdin i=dat1.csv o=rsl6.csv
$ more rsl6.csv
customer,date,amount,fileName
A,20081201,10,dat1.csv
B,20081002,40,dat1.csv
```

Example 7: Specify wild card

Specifying wild card `dat* .csv` to concatenate the three CSV files `dat1 .csv, dat2 .csv, dat3 .csv` in the current directory.

```
$ more dat1 .csv
customer,date,amount
A,20081201,10
B,20081002,40
$ more dat2 .csv
customer,date,amount
A,20081207,20
A,20081213,30
B,20081209,50
$ more dat3 .csv
customer,date,quantity
A,20081201,3
B,20081002,1
$ mcat -force i=dat*.csv o=rsl7.csv
#END# kgcat -force i=dat*.csv o=rsl7.csv
$ more rsl7.csv
customer,date,amount
A,20081201,10
B,20081002,40
A,20081207,20
A,20081213,30
B,20081209,50
A,20081201,3
B,20081002,1
```

Example 8: Concatenate the same file multiple times

Same file can be specified more than one time.

```
$ mcat i=dat1 .csv, dat1 .csv, dat1 .csv o=rsl8 .csv
#END# kgcat i=dat1 .csv, dat1 .csv, dat1 .csv o=rsl8 .csv
$ more rsl8 .csv
customer,date,amount
A,20081201,10
B,20081002,40
A,20081201,10
B,20081002,40
```

Related command

`msep` : Reverse the operation mentioned above and separate data files.
3.8  mchgnum - Substitute Values within Numerical Range

The field name for encoding is specified at f= parameter, number and range criteria is specified at the R= parameter, the substitution string specified in the v= parameter replaces the value in the defined field.

Format


Parameters

f= Replace the specified field (multiple fields can be specified) according to the replacement string list at v= parameter and the numerical ranges list at R= parameter.
R= Specify the numerical range to be replaced (multiple fields can be specified)
1.1,2.5 : more than 1.1 and less than 2.5).
Use MIN for minimum value, MAX for maximum value ( MIN, 2.5 : 2.5 or less).
O= Out of range character strings
Specify the replacement string when values fall outside the numeric range list at the R= parameter (returns NULL values in output when this parameter is not specified).
-F Display out of range values in the column.
Retain the out of range values in the output even though the values fall outside the specified numeric range defined in the R= parameter.
-v= Specify the replacement character string corresponding to the numerical range in the R= parameter.
More than 1 argument must be defined at R=.
-A This option adds output to a new column instead of replacing the specified item.
-r The range defined at R= parameter deals with ' greater than less than'.
For example, 1.1_2.5 represents 'greater than 1.1 less than 2.5'.

Examples

Example 1: Basic Example
Encodes the numeric values in quantity column to character strings where values of less than but not equal to 10 are treated as low, 10 or more but less than 20 are treated as middle, values of 20 or more is treated as high.

```
$ more dat1.csv
customer,quantity
A,5
B,10
C,15
D,2
E,50

$ mchgnum f=quantity R=MIN,10,20,MAX v=low,middle,high i=dat1.csv o=rsl1.csv
```

Example 2: Equal to parameter range
Replace the numeric values in quantity column to character strings where 10 or below is treated as low, more than 10 but less than or equal to 20 is treated as middle, values of 20 or more is treated as high.

```
$ more dat1.csv
customer,quantity
A,5
B,10
C,15
D,2
E,50

$ mchgnum f=quantity R=MIN,10,20,MAX v=low,middle,high i=dat1.csv o=rsl1.csv
```

```
$ more rsl1.csv
customer,quantity
A,low
B,middle
C,middle
D,low
E,high
```
Example 3: Replace values out of the list of numeric range

Replace numeric values in `quantity` column to character strings where 10 or above and less than 20 is coded as `low`, 20 or above and less than 30 is coded as `middle`, 30 or more is coded as `high`, values that are less than 10 is coded as `out of range`.

Example 4: Add a new column

Replace the numeric values in `quantity` column to character strings where values less than 10 is treated as `low`, 10 or above but less than 20 is treated as `middle`, 20 or above is treated as `high`. Store the output of replacement strings in a new column as `evaluate`.

Example 5: Display original values in column if out of defined range

Replace the numeric values in `quantity` column to character strings where values of 10 or above but less than 20 is coded as `low`, 20 or above but less than 30 is coded as `middle`, 30 or above is coded as `high`. Retain original values in the output if the value is less than 10.

Related Commands

`mchgstr` : Substitute string.

`msed` : Replace text with regular expressions.
3.9 mchgstr - Substitute String

Replace the field(s) specified in the f= parameter with the string according to the replacement criteria specified in the c= parameter.

Format

mchgstr c= f= [0=] [-A] [-F] [-sub] [-W] [i=] [o=] [nfn] [nfno] [-x] [--help] [--version]

Parameters

- **c=** Specify the replacement character string to be replaced.
- **f=** Specify the field(s) (multiple fields can be specified) where the character string is replaced according to the replacement criteria specified in the *c=* parameter.
- **O=** Specify the replacement string if the values does not match any substitution criteria specified in the *c=* parameter (returns NULL values when this parameter is not specified).
- **-A** This option adds output in a new column instead of replacing the specified item.
- **-F** Retain values in output even though values fall outside the specified numeric range defined in the R= parameter.
- **-sub** Compare using substring match rather than exact match.
  Search the values specified in the *f=* parameter, and replace the string with the criteria specified in *c=*.
- **-W** Match wide-character substring with -sub option.

Examples

Example 1: Basic Example

Replace values in the column from "01" to "A", "03" to "B", "04" to "C". Other values that do not match the criteria are returned as NULL values in the output.

```
$ more dat1.csv
id,item
1,01
2,02
3,03
4,04
5,05
$ mchgstr f=item c=01:A,03:B,05:C i=dat1.csv o=rsl1.csv
#END# mchgstr c=01:A,03:B,05:C f=item i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,item
1,A
2,
3,B
4,
5,C
```

Example 2: Replace values outside the criteria

Use the 0= parameter to replace character string that do not match the substitution criteria to "out of range" in the output.

```
$ mchgstr f=item c=01:A,03:B,05:C 0="out of range" i=dat1.csv o=rsl2.csv
#END# mchgstr 0="out of range" c=01:A,03:B,05:C f=item i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,item
1,A
2,out of range
3,B
4,out of range
5,C
```
Example 3: Example 3: Add new column in output

Define the name of new column (item info) in output with -A option.

```bash
$ mchgstr f=item:"item info" c=01:A,03:B,05:C O="out of range" -A i=dat1.csv o=rsl3.csv
#END# kgchgstr -A O=out of range c=01:A,03:B,05:C f=item:item info i=dat1.csv o=rsl3.csv

$ more rsl3.csv
id,item,item info
1,01,A
2,02,out of range
3,03,B
4,04,out of range
5,05,C
```

Example 4: Display original value in field falls outside the criteria

When -F option is specified, output value of the field remains the same if it does not match any of the substitution criteria.

```bash
$ mchgstr f=item c=01:A,03:B,05:C -F i=dat1.csv o=rsl4.csv
#END# kgchgstr -F c=01:A,03:B,05:C f=item i=dat1.csv o=rsl4.csv

$ more rsl4.csv
id,item
1,A
2,02
3,B
4,04
5,C
```

Example 5: Replace matching substrings

Replace substring with -sub option specified. In following example, values where item field contains "01" will be substituted with "A".

```bash
$ more dat2.csv
id,item
1,0111
2,0121
3,0231
4,0241
5,0151
$ mchgstr f=item c=01:A -sub i=dat2.csv o=rsl5.csv
#END# kgchgstr -sub c=01:A f=item i=dat2.csv o=rsl5.csv

$ more rsl5.csv
id,item
1,A11
2,A21
3,
4,
5,A51
```

Example 6: Wide character substring match

Use the option -W to replace wide-characters strings. However, if you are using UTF-8 encoding, it is not necessary to define -W. Refer to the section "Multibyte characters" for details.

```bash
$ more dat3.csv
id,city
1,奈良市
2,下市町
3,上津川村
4,五條市
5,山添村
$ mchgstr f=city c=市:01,町:02,村:02 -sub -W i=dat3.csv o=rsl6.csv
#END# kgchgstr -W -sub c=市:01,町:02,村:02 f=city i=dat3.csv o=rsl6.csv

$ more rsl6.csv
id,city
1,奈良市
```
Related Commands

- **mchgnum**: Substitution based on numeric range.
- **msed**: Replace string by regular expression.
3.10 mchkcsv - Check CSV Data

Automatically repair (standardize the number of fields for all rows) comma-separated-values (CSV) data that do not meet the format for MCMD. Use the `--diag` option to perform checking on CSV data.

**Format**

```
mchkcsv [a=] [-diag] [-r] [i=] [o=] [-nf] [-nfn] [--help] [--version]
```

**Parameters**

- `i=`: Input file name
  The CSV data defined here is checked for incomplete lines. Issues identified in this file is automatically repaired. If this parameter is omitted, the standard input is used.
- `a=`: Substitute the field name(s) in the input data with the field name(s) specified at this parameter.
  If the number of fields specified is less than the number of fields in the header, the output will only yield portion of columns from the input data starting from the left. Conversely, if the number of fields specified here is greater than the number of fields in input data, data items in the extra fields are expressed as NULL values in the output.
- `--diag`: Execute check. When this option is specified, write result to standard output.
- `--r`: Exclude control characters
  This defines the control character as ASCII character code 0x00-0x1f,0x7f (remove 0x09,0x0a,0x0d). Control code is automatically converted to the string `&amp;#x` when this option is not specified.

**Examples**

**Example 1: Repair data**

This data contains different number of columns in all records. For instance, only 3 records have 4 columns. Use M-Command to repair and standardize 3rd and 5th rows to 4 columns.

```
$ more dat1.csv
product,date,quantity,amount
A,20081201,1,10
A,20081202,2,1
A,*,3
B,20081201,4,40
B,20081203,50

$ mchkcsv i=dat1.csv o=rsl1.csv
#END# kgchkcsv i=dat1.csv o=rsl1.csv
$ more rsl1.csv
product,date,quantity,amount
A,20081201,1,10
A,20081202,2,1
A,*,3,
B,20081201,4,40
B,20081203,50,
```

**Example 2: Change field name after repairing the data**

This data contains different number of columns in all records. For instance, only 3 records have 4 columns. Use M Command to repair and standardize 3rd and 5th rows to 4 columns. At the same time, label the field names from the input data as `verb—[] product,date,quantity,amount []—` starting from the left.

```
$ more dat2.csv
fld1,fld2,fld3,fld4
A,20081201,1,10
A,20081202,2,1

$ mchkcsv i=dat2.csv o=rsl2.csv
#END# kgchkcsv i=dat2.csv o=rsl2.csv
$ more rsl2.csv
product,date,quantity,amount
A,20081201,1,10
A,20081202,2,1
A,*,3
B,20081201,4,40
B,20081203,50,
```
Example 3: Check data integrity and output diagnostic results

It merely checks for incomplete data structure in the CSV data, and save the diagnosis result in CSV file.

```bash
$ mchkcsv -diag i=dat1.csv o=rsl3.csv
#END# kgchkcsv -diag i=dat1.csv o=rsl3.csv
$ more rsl3.csv
```

```
#===================================================
# CSV file name : dat1.csv
#---------------------------------------------------
# # : KGMOD
# ? : OK
# # 1 product
# 2 date
# 3 quantity
# 4 amount
# # "EOL(End Of Line) 情報 (ヘッダー含む)"
# 6 (LineNo: 0 1 2 ... )
# # データ行情報 (ヘッダー含まない)
# 総行数 : 5
# 総バイト数 : 66
# 平均長 : 13.2
# 最大長 : 16 (LineNo:2)
# 最小長 : 6 (LineNo:4)
# 情報 : 項目数の一貫性
# 項目番号 [1] 項目名 [product]
# NULL 値の行数 : 0
# DQUOTE で囲われていない行数 : 5 (LineNo: 1 2 3 ... )
# DQUOTE で囲われている行数 : 0
# 項目番号 [2] 項目名 [date]
# NULL 値の行数 : 0
# DQUOTE で囲われていない行数 : 5 (LineNo: 1 2 3 ... )
# DQUOTE で囲われている行数 : 0
# 項目番号 [3] 項目名 [quantity]
# NULL 値の行数 : 0
# DQUOTE で囲われていない行数 : 5 (LineNo: 1 2 3 ... )
# DQUOTE で囲われている行数 : 0
# 項目番号 [4] 項目名 [amount]
# NULL 値の行数 : 1 (LineNo: 2 )
```
### 3. COMMAND REFERENCE

<table>
<thead>
<tr>
<th>DQUOTE で囲まれていない行数</th>
<th>3 (LineNo: 124)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQUOTE で囲まれている行数</td>
<td>0</td>
</tr>
</tbody>
</table>

#### 問題点の解説

- **?a:** 同じ項目名があると項目番号を特定できない。
  - 【対処方法】kgchkcsv a=x,y,z のように項目名を新たに指定する。
- **?b:** 項目名に不正な文字があるとエラーになる。
  - 【対処方法】kgchkcsv a=x,y,z のように項目名を新たに指定する。
- **?c:** KGMOD が扱う行は高度化のため LF(UNIX 改行) のみ。
  - この問題は RFC4180 には準拠しておらず KGMOD 独自の制約である。
  - 【対処方法】kgchkcsv にて全て LF に変換される。
- **?d:** 最終行に LF や CR などの改行 (EOL) 文字が存在しない。
  - これは RFC4180 にも準拠していない。
  - 【対処方法】kgchkcsv にて LF が付加される。
- **?e:** データファイル内に:\n が入り込んでいる。
  - テキストファイルでない可能性が高い。
  - RFC4180 には準拠していない。
  - 【対処方法】kgchkcsv にて、"&#x00"
    - kgchkcsv -r にて \n は削除される。
- **?f:** KGMOD が扱う一行の最大長を超過している。
  - 現在の設定では 1024000 バイト以上の長さの行は扱えない。
  - 【対処方法】環境変数を設定することで最大値を変更可能である。
    - ex) export KG_MaxRecLen=204800
      - ただし 1024000 バイトを超えては指定できない。
      - この問題は RFC4180 には準拠しており KGMOD 独自の制約である。
- **?g:** KGMOD では全行同じ項目数を前提とする。
  - この問題は RFC4180 には準拠しており KGMOD 独自の制約である。
  - 【対処方法】
    1) kgchkcsv データ HEADER の項目数に合わせる。
    2) kgchkcsv a=x,y,z HEADER 行をスキップし、
      指定した x,y,z を項目名として 1) と同様の処理を行う。
- **?h:** 制御文字 (0x01~0x1F,0x7F) が項目値として入り込んでいる。
  - テキストファイルでない可能性が高い。
  - RFC4180 には準拠していない。
  - 【対処方法】kgchkcsv にて、"&#x01"
    - kgchkcsv -r にて制御文字は削除される。
- **?i:** TAB は利用できない。
  - RFC4180 には準拠していない。
  - 【対処方法】kgchkcsv にて、"&#x09"
    - kgchkcsv -r にて TAB は削除される。
- **?j:** DQUOTE で囲まれていない中で DQUOTE が見つかった
  - RFC4180 には準拠していない。
  - 【対処方法】kgchkcsv にて上記の変換を行う。
- **?k:** DQUOTE で囲まれている中で单一の DQUOTE が見つかった
  - RFC4180 には準拠していない。
  - 【対処方法】kgchkcsv にて上記の変換を行う。

---

Related Command
3.11 mcombi - Compute Combination

The `mcombi` command is used to compute combinations of fields.

**Format**

```
mcombi a= f= n= [s=] [k=] [-p] [-dup] [i=] [o=] [-nfn] [-nfno] [-x] [-q] [--help] [--version]
```

**Parameters**

- **a=**: Name of the field to be added.
- **f=**: Compute the combinations for the set of field name(s) (multiple fields can be specified) specified.
- **n=**: Number of combinations.
  - When you increase the number of combinations, note that the number of output records will increase exponentially.
- **s=**: After sorted by specified field (multiple fields can be specified), combinations of items specified in field `f=` are enumerated.
- **k=**: List of key field name(s) (multiple fields can be specified)
  - Compute combinations based on the list of key field name(s).
- **-p**: Compute the permutations.
- **-dup**: Output combinations with the same value.

**Examples**

**Example 1: Basic Example**

Enumerate all combinations of two items in the `item` field for each `customer`, and save the output in `item1,item2`. Fields not specified at `k=,f=` (item field in this case) remains after the key field column.

```bash
$ more dat1.csv
customer,item
A,a1
A,a2
A,a3
B,a4
B,a5
$ mcombi k=customer f=item n=2 a=item1,item2 i=dat1.csv o=rsl1.csv
#END# kgcombi a=item1,item2 f=item i=dat1.csv k=customer n=2 o=rsl1.csv
$ more rsl1.csv
customer%0,item,item1,item2
A,a3,a1,a1
A,a3,a1,a2
A,a3,a1,a3
A,a3,a2,a2
A,a3,a2,a3
A,a3,a3,a3
B,a5,a4,a4
```

**Example 2: Basic Example 2**

When you specify the `-dup` option, the output includes combination of the same field.

```bash
$ mcombi k=customer f=item n=2 a=item1,item2 i=dat1.csv o=rs12.csv -dup
#END# kgcombi -dup a=item1,item2 f=item i=dat1.csv k=customer n=2 o=rs12.csv
$ more rs12.csv
customer%0,item,item1,item2
A,a3,a1,a1
A,a3,a1,a2
A,a3,a1,a3
A,a3,a2,a2
A,a3,a2,a3
A,a3,a3,a3
B,a5,a4,a4
```
Example 3: Compute permutation

Enumerate permutation of two items in the \texttt{item} field for each \texttt{customer}, and save the output in column \texttt{item1,item2}.

\begin{verbatim}
$ mcombi k=customer f=item n=2 a=item1,item2 -p i=dat1.csv o=rsl3.csv
#END# kgcombi -p a=item1,item2 f=item i=dat1.csv k=customer n=2 o=rsl3.csv
$ more rsl3.csv
customer%0,item,item1,item2
A,a3,a1,a2
A,a3,a2,a1
A,a3,a1,a3
A,a3,a3,a1
A,a3,a2,a3
A,a3,a3,a2
B,a5,a4,a5
B,a5,a5,a4
\end{verbatim}

Related Command
### 3.12 mcommon - Select Common Records in Reference File

Compare records in the input file with the ones in the reference file, at which reference file is specified in \texttt{m=} parameter. Set the key parameter at \texttt{k=} for selecting records common in both files.

#### Format

\[
\text{mcommon } \text{ k=} [\text{K=}] [\text{u=}] [-r] \text{ m=} \text{ i=} \text{ [o=} \text{-nfn} \text{-nfnfno} \text{-x} \text{-q} \text{ [--help]} \text{ [--version]}
\]

#### Parameters

- \texttt{k=} List of key field(s) to match the input data (Multiple keys can be specified). Records in the input file matching the key field(s) in the reference file specified at \texttt{K=} is selected.
- \texttt{m=} Specify reference file name.
- \texttt{K=} Key field(s) to match with from the reference data (Multiple keys can be specified). Records in the reference file that matches the key field specified in the \texttt{k=} parameter is selected. The parameter is not required if the key field is the same on both input data and reference file. Records with same key values will therefore appear in consecutive rows.
- \texttt{u=} File name of output with unmatched records.
- \texttt{-r} Reverse selection. Compare the value specified at \texttt{k=} parameter from the input file with the value from the reference file specified at \texttt{m=} parameter, and select unmatched record(s) from the input file.

#### Examples

**Example 1: Basic Example**

Select records with common customers in both input file and reference file. Save unmatched records to a separate file \texttt{oth.csv}.

```bash
$ more dat1.csv
Customer,Quantity
A,1
B,2
C,1
D,3
E,1
$ more ref1.csv
Customer,Gender
A,Female
B,Male
E,Female
$ mcommon k=Customer m=ref1.csv u=oth.csv i=dat1.csv o=rsl1.csv
#END# kgcommon i=dat1.csv k=Customer m=ref1.csv o=rsl1.csv u=oth.csv
$ more rsl1.csv
Customer%0,Quantity
A,1
E,1
$ more oth.csv
Customer%0,Quantity
C,1
D,3
```

**Example 2: Select unmatched records from the input file**

Reverse selection criteria by using the \texttt{-r} option, the "Customer" not in the reference file are selected.
Example 3: Different names of join key

If the join key field name in the reference file is different, specify the field name at `¥ verb—K=—`.  

Example 4: Example with duplicate key fields

Record selection with duplicate records in both input file and reference file.

Related commands

`msselstr`: This command can be used when the types of join key in reference file is not a lot.  
`mnrcommon`: This command can be used when the join key in the reference file is not unique.  
`mjoin`: This command is not only used for selecting data, but also for combining fields.
3.13 mcount - Count the Number of Rows

Count the number of rows and store the results in a new column defined in a= parameter. Counting is carried out by each aggregate key when k= parameter is specified. Otherwise, if k= is not specified, all rows are counted.

Format

mcount a= [k=] [i=] [o=] [-nfn] [-nfno] [-x] [-q] [--help] [--version]

Parameters

a= Specify the new field name to be added.
   This parameter is not required when nfn option is specified.

k= Key item(s) (Multiple keys can be specified) [aggregate key break processing]
   Count the number of instances for incremental rows based on the key field(s) defined.

Examples

Example 1: Basic Example

Count the number of rows by date, and save the results in a new column count.

```
$ more dat1.csv
date
20090109
20090109
20090109
20090110
20090110
$ mcount k=date a=count i=dat1.csv o=rsl1.csv
#END# kgcount a=count i=dat1.csv k=date o=rsl1.csv
$ more rsl1.csv
date, count
20090109,3
20090110,2
```

Example 2: Count without aggregate key

Count the number of rows without specifying the aggregate key.

```
$ mcount a=count i=dat1.csv o=rsl2.csv
#END# kgcount a=count i=dat1.csv o=rsl2.csv
$ more rsl2.csv
date, count
20090109,3
20090110,2
```

Related command

mstats: Specify c=count to count non-null values in the data.
3.14  mcross - Crosstab

Build a crosstab. Transpose the fields specified at s= parameter horizontally, itemize the array of data specified at the f= parameter to the corresponding key specified at k=.

Format

mcross f= s= [a=] [k=] [v=] [i=] [o=] [-nf] [-nfn] [-nfno] [-x] [-q] [--help] [--version]

Parameters

f=  Field name specified will become cross tab fact cell.
   When multiple fields are specified, the variables will be transposed to multiple rows.
   The fld field is created to identify multiple rows of the field.
   Specify the name of field at f= as the column variable.
   Use a= parameter to rename the field fld.

s=  Field names of transposed field.
   The data series in the field becomes column name of transposed rows.
   a=  Specify the field name to transpose the row of field names.
   The field name is set to fld by default when this parameter is not defined.
   k=  Key field name [aggregate key break processing]
   Key to transpose data into horizontal rows.
   v=  Null value replacement string
   Replace null character with a character defined at v= parameter.

Example

Example 1: Basic Example

Expand the array of date horizontally and itemize quantity to the corresponding item.

```bash
$ more dat1.csv
item,date,quantity,price
A,20081201,1,10
A,20081202,2,20
A,20081203,3,30
B,20081201,4,40
$ mcross k=item f=quantity s=date i=dat1.csv o=rsl1.csv
#END# kgcross f=quantity i=dat1.csv k=item o=rsl1.csv s=date
$ more rsl1.csv
item%0,fld,20081201,20081202,20081203
A,quantity,1,2,3
B,quantity,4,,5
```

Example 2: Restore the original input data

Restore the output from Example 1 to the original input data with mcross.

```bash
$ more rsl1.csv
item%0,fld,20081201,20081202,20081203
A,quantity,1,2,3
B,quantity,4,,5
$ mcross k=item f=2008* s=fld i=rsl1.csv o=rsl2.csv
#END# kgcross f=2008* i=rsl1.csv k=item o=rsl2.csv s=fld
$ more rsl2.csv
item%0,date,quantity
A,20081201,1
A,20081202,2
A,20081203,3
B,20081201,4
```
Example 3: Crosstab with multiple fields

Display crosstab results on two fields quantity, price.

```
$ mcross k=item f=quantity,price s=date i=dat1.csv o=rsl3.csv
#END# kgcross f=quantity,price i=dat1.csv k=item o=rsl3.csv s=date
$ more rsl3.csv
item%0,fld,20081201,20081202,20081203
A,quantity,1,2,3
A,price,10,20,30
B,quantity,4,,5
B,price,40,,50
```

Example 4: Reverse data sequence

Restore the sequence of the items that was expanded horizontally.

```
$ mcross k=item f=quantity,price s=date%r i=dat1.csv o=rsl4.csv
#END# kgcross f=quantity,price i=dat1.csv k=item o=rsl4.csv s=date%r
$ more rsl4.csv
item%0,fld,20081203,20081202,20081201
A,quantity,3,2,1
A,price,30,20,10
B,quantity,5,,4
B,price,50,,40
```

Related Command

mtra: Creates the same data image of horizontal transposition, but mtra output the vector as a single field in the output.
3.15 mcsv2arff - Convert csv to arff Format

Convert csv formatted data into arff file (data format for WEKA). User must specify the type of attribute for arff, for instance, \texttt{d=} defines category format field, \texttt{n=} defines numeric format field, \texttt{s=} defines string format field, and finally, \texttt{D=} defines date format field. The date format includes time information when attached \%t to date format field name.

**Format**

\texttt{mcsv2arff n=}\texttt{|d=}\texttt{|D=}\texttt{|s=} [\texttt{T=}] \texttt{i=} [\texttt{o=}] [\texttt{--help}] [\texttt{--version}]

**Parameters**

\texttt{n=} Numeric field name(s) (multiple items can be specified).
\texttt{d=} Category field name(s) (multiple items can be specified).
\texttt{D=} List of date (time) field name(s) (multiple items can be specified). \texttt{[\%t]}
  - When \%t is not specified: yyyyMMdd
  - When \%t is specified: yyyyMMddHHmmss
\texttt{s=} Character string field names (multiple items can be specified).
\texttt{T=} Title in character string.

**Examples**

**Example 1: Convert csv format data to arff format**

Convert data to arff format and define "customer" field as string type, "product" field as category type, "date" field as date type (exclude time), \texttt{quantity} and \texttt{amount} fields as numeric attributes.

\begin{verbatim}
$ more dat1.csv
customer,product,date,quantity,amount
No.1,A,20081201,1,10
No.2,A,20081202,2,20
No.3,A,20081203,3,30
No.4,B,20081201,4,40
No.5,B,20081203,5,50
$ mcsv2arff s=customer d=product D=date n=quantity,amount T=Customer_Purchase_Data i=dat1.csv o=rsl1.csv
$ more rsl1.csv
@RELATION Customer_Purchase_Data
@ATTRIBUTE customer string
@ATTRIBUTE date date yyyyMMdd
@ATTRIBUTE quantity numeric
@ATTRIBUTE amount numeric
@ATTRIBUTE product \{A,B\}
@DATA
No.1,20081201,1,10,A
No.2,20081202,2,20,A
No.3,20081203,3,30,A
No.4,20081201,4,40,B
No.5,20081203,5,50,B
\end{verbatim}

**Example 2: Convert csv format data to arff format (include time in the date attribute)**

Specify the date with the time information by adding \%t such that \texttt{D=}\texttt{date\%t}.

\begin{verbatim}
$ more dat2.csv
customer,product,date,quantity,amount
No.1,A,20081201102030,1,10
No.2,A,20081202123010,2,20
No.3,A,20081203153010,3,30
$ mcsv2arff D=date%t T=Customer_Purchase_Data d=product i=dat2.csv n=quantity,amount o=rsl2.csv s=customer
$ more rsl2.csv
@RELATION Customer_Purchase_Data
@ATTRIBUTE customer string
@ATTRIBUTE date date yyyyMMdd
@ATTRIBUTE quantity numeric
@ATTRIBUTE amount numeric
@ATTRIBUTE product \{A,B\}
@DATA
No.1,20081201102030,A
No.2,20081202123010,A
No.3,20081203153010,A
No.4,20081201401010,B
No.5,20081203501010,B
\end{verbatim}
### 3.15. MCSV2ARFF - CONVERT CSV TO ARFF FORMAT

```
No.4,B,20081201174010,4,40
No.5,B,20081203133010,5,50

$ mcsv2arff s=customer d=product D=date%t n=quantity,amount T=Customer_Purchase_Data i=dat2.csv o=rsl2.csv

#END# kgcsv2arff D=date%t T=Customer_Purchase_Data d=product i=dat2.csv n=quantity,amount o=rsl2.csv s=customer

$ more rsl2.csv

@RELATION Customer_Purchase_Data

@ATTRIBUTE customer string
@ATTRIBUTE date date yyyyMMddHHmmss
@ATTRIBUTE quantity numeric
@ATTRIBUTE amount numeric
@ATTRIBUTE product {A,B}

@DATA
No.1,20081201102030,1,10,A
No.2,20081202123010,2,20,A
No.3,20081203153010,3,30,A
No.4,20081201174010,4,40,B
No.5,20081203133010,5,50,B
```

**Related Command**

**marff2csv**: Reverse conversion.

**Reference**

3.16 mcut - Select Column

Extract the specified column(s). The specified column is removed with -r option.

Format

mcut f= [-r] [i=] [o=] [-nfn] [-nfno] [-x] [--help] [--version]

Parameters

- **f=** Define name of column to be extracted
  
  New column name for can be specified by defining the field name, followed by colon and the new field name.
  
  ex. f=a:A,b:B

- **-r** Field removal switch
  
  Remove all columns defined in the f= parameter.

- **-nfn** When header is not present in first row of the input data, position number of column is used to identify corresponding field(s).
  
  New column name(s) for each column can be specified in the output file as follows.
  
  Example f=0:date,2:store,3:quantity

Examples

Example 1: Basic Example

Extract customer and amount information from the data file `dat1.csv` Rename the column ”amount ” to ”sales” in the output.

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20

$ mcut f=customer,amount:sales i=dat1.csv o=rsl1.csv
#END# kgcut f=customer,amount:sales i=dat1.csv o=rsl1.csv

$ more rsl1.csv
customer,sales
A,10
A,20
B,15
B,10
B,20
```

Example 2: Remove columns

Remove columns customer and amount specified at -r.

```
$ mcut f=customer,amount -r i=dat1.csv o=rs22.csv
#END# kgcut -f=customer,amount i=dat1.csv o=rs22.csv

$ more rs22.csv
quantity
1
2
3
1
```
Example 3: Data without field names

Select columns 0, 2 from an input file without field header, add customer and amount as field names in the output file.

```bash
$ mcut f=0:customer,2:amount -nfni i=dat1.csv o=rsl3.csv
#END# kgcut -nfni f=0:customer,2:amount i=dat1.csv o=rsl3.csv
$ more rsl3.csv
customer,amount
customer,amount
A,10
A,20
B,15
B,10
B,20
```

related command

`mfldname`: Use `mfldname` to change the field names.
3.17 mdata - Generate Datasets

Generate a variety of data sets, including the best known Iris data set in the field of pattern recognition and data sets for the MCMMD tutorial. Noted that the way of defining parameters in this command is completely different with other commands.

Format

mdata dataset/parameter1/parameter2/…

Parameters

Define the the name of data set and parameter separated with "/". The dataset summary is listed under contents in Table 3.1. The usage of parameter differs depending on the dataset, its corresponding details of the parameters for each data set can be found in the Table below.

<table>
<thead>
<tr>
<th>Name of data set</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>iris</td>
<td>This dataset records characteristics and size of sepal and petal of a variety of Iris best for building classification model.</td>
<td>N/A</td>
</tr>
<tr>
<td>man0</td>
<td>Data used in Figure 2.1 of this manual</td>
<td>N/A</td>
</tr>
<tr>
<td>man1</td>
<td>Data used in Figure 2.3 of this manual</td>
<td>N/A</td>
</tr>
<tr>
<td>tutorial_jp</td>
<td>Artificial supermarket purchasing data is used. Comprised of customer master, product master, and multiple files.</td>
<td>If the data name is specified, mdata writes the specified data to the standard output. If not specified, all files will be generated under the directory tutorial_jp. The details of each data name are as follows. dat: Purchasing data syo: Product master cust: Customer master jicfs1, jicfs2, jicfs4, jicfs6: Product category master</td>
</tr>
<tr>
<td>tutorial_en</td>
<td>English version of the tutorial_jp dataset</td>
<td>Same as tutorial_jp</td>
</tr>
</tbody>
</table>

Examples

Example 1 Generate iris dataset

Write Iris dataset to standard output.

```
$ mdata iris
SepalLength,SepalWidth,PetalLength,PetalWidth,Species
5.1,3.5,1.4,0.2,setosa
4.9,3,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa
```

Example 2 Create Tutorial dataset

Create all files for the tutorial dataset.

```
$ mdata tutorial_en
#END mdata tutorial_en
$ ls -1 tutorial_en
total 4704
-rw-r--r-- 1 hamuro staff 20673 8 22 08:14 cust.csv
-rw-r--r-- 1 hamuro staff 2281312 8 22 08:14 dat.csv
```
Example 3 Create individual tutorial dataset

Write Product Master dataset to standard output.

```bash
$ mdata tutorial_en/syo

product,productName,Code1Desc,Code2Desc,Code4Desc,Code6Desc,manufacturer,brand,unitCost
0000000,Processed_Seafood_2,1,11,1197,119705,0495,049502,310
0000001,Other_Yogurt_Drink_2,1,14,1404,140497,1658,165801,215
0000002,Carbonic_Flavor_3,1,14,1403,140305,1911,191100,406
```

```bash
$ more tutorial_en/dat.csv

<table>
<thead>
<tr>
<th>customer</th>
<th>dob</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000A</td>
<td>19711107</td>
<td>female</td>
</tr>
<tr>
<td>00000B</td>
<td>19461025</td>
<td>female</td>
</tr>
<tr>
<td>00000C</td>
<td>19660307</td>
<td>female</td>
</tr>
</tbody>
</table>

```
3.18 mdelnull - Remove rows with null values

Remove the row if values defined at \texttt{f=} parameter contain null value(s).

**Format**

```
mdelnull f= [k=] [u=] [-F] [-r] [-R] [i=] [o=] [bufcount=} [-fnn] [-nfno] [-x] [-q] [--help] [--version]
```

**Parameters**

- \texttt{f=}  Search NULL values in specified field name(s) (Multiple fields can be specified).
- \texttt{k=}  Remove records with null values based on the same key field(s) defined (multiple fields can be specified).
- \texttt{u=}  Print unmatched records to the output file specified.
- \texttt{-F}  AND condition for multiple fields
  
  Records consisting of null values in multiple fields defined at \texttt{f=} parameter will be selected and removed.
- \texttt{-r}  Reverse selection
  
  Selected records are not removed.
- \texttt{-R}  AND condition for multiple records
  
  Remove all records if the field with the same key field specified at the \texttt{k=} parameter contains NULL values.

**Example**

**Example 1: Basic Example**

Remove records where \texttt{Quantity} and \texttt{Amount} contain null values. Save records with null values to a separate file \texttt{oth.csv}.

```bash
$ more dat1.csv
Customer,Quantity,Amount
A,1,10
A,,20
B,1,15
B,3,
C,,20
$ mdelnull f=Quantity,Amount u=oth.csv i=dat1.csv o=rsl1.csv
#END# kgdelnull f=Quantity,Amount i=dat1.csv o=rsl1.csv u=oth.csv
$ more rsl1.csv
Customer,Quantity,Amount
A,1,10
B,1,15
C,,20
$ more oth.csv
Customer,Quantity,Amount
A,,20
B,3,
```

**Example 2: Select rows with NULL values**

Select records with NULL values by specifying \texttt{-r}.

```bash
$ mdelnull f=Quantity,Amount -r i=dat1.csv o=rsl2.csv
#END# kgdelnull -r f=Quantity,Amount i=dat1.csv o=rsl2.csv
$ more rsll2.csv
Customer,Quantity,Amount
A,,20
B,3,
```
Example 3: Remove records with the same key if any record contains NULL values

Remove based on the aggregate key specified at k=. Remove records where Quantity and Amount contain null values for each customer.

```
$ mdelnull k=Customer f=Quantity,Amount i=dat1.csv o=rsl3.csv
#END# kgdelnull f=Quantity,Amount i=dat1.csv k=Customer o=rsl3.csv
$ more rsl3.csv
Customer%0,Quantity,Amount
C,1,20
```

Example 4: AND condition between fields

Remove the record where both Quantity and Amount fields contain null values.

```
$ more dat2.csv
Customer,Quantity,Amount
A,1,10
A,,15
B,1,15
B,3,
C,1,20
$ mdelnull f=Quantity,Amount -F i=dat2.csv o=rsl4.csv
#END# kgdelnull -F f=Quantity,Amount i=dat2.csv o=rsl4.csv
$ more rsl4.csv
Customer,Quantity,Amount
A,1,10
B,1,15
B,3,
C,1,20
```

Example 5: AND condition between records

Remove the Customer record if all values in Quantity is NULL.

```
$ mdelnull k=Customer f=Quantity -R i=dat1.csv o=rsl5.csv
#END# kgdelnull -R f=Quantity i=dat1.csv k=Customer o=rsl5.csv
$ more rsl5.csv
Customer%0,Quantity,Amount
A,1,10
A,,20
B,1,15
B,3,
C,1,20
```

Related Command

`mnullto`: NULL value(s) in records are convert to specified character strings.

Does not delete the row that contains a NULL value, it is converted to a string of digits of a NULL value.
CHAPTER 3. COMMAND REFERENCE

3.19 mdformat Extract Date Time

CSV data that was exported from other systems often contains forward slash and colon symbol in date columns, in addition, date and time are stored in single digit (Example: 2014/7 18:1:57). Sorting and range specification processing on these kind of items in MCMD is not possible.

For ease of processing data and time formatted data **Date Time Format**, the **mdformat** command extracts the date, hour, minute, and second on fields specified in `f=` parameter according to the format specified in the `c=` parameter.

**Format**

```
mdformat c= f= [-A] [i=] [o=] [-nfn] [-nfno] [-x] [--help] [--version]
```

**Parameters**

- `f=` Specify the field name for extraction (multiple items can be specified).
- `c=` Specify string format according to the specified format.
- `-A` Specify the new column name and save output results in new column.

**Conversion Specification of Character Format**

Table 3.2 shows the possible character conversion formats that can be defined in `c=` parameter.

<table>
<thead>
<tr>
<th>Conversion Characters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Y</td>
<td>Number representing year (4 digits)</td>
</tr>
<tr>
<td>%y</td>
<td>Number representing year (2 digits)</td>
</tr>
<tr>
<td>%m</td>
<td>Number representing month (2 digits)</td>
</tr>
<tr>
<td>%d</td>
<td>Number representing day (2 digits)</td>
</tr>
<tr>
<td>%H</td>
<td>Time (2 digits)</td>
</tr>
<tr>
<td>%M</td>
<td>Minute (2 digits)</td>
</tr>
<tr>
<td>%S</td>
<td>Second (2 digits)</td>
</tr>
</tbody>
</table>

**Examples**

**Example 1: Basic Example**

Extract and convert time and date information from `fld` field. Save the converted format as "a:yearmonthday:b:timeminutesecond" by specifying "a:%Y%m%d:b:%H%M%S" in the `c=` parameter.

```
$ more dat1.csv
fld
a:20120304:b:121212
a:20101204:b:011309
$ mdformat f=fld c=a:%Y%m%d:b:%H%M%S i=dat1.csv o=rsl1.csv
#END# kgdformat c=a:%Y%m%d:b:%H%M%S f=fld i=dat1.csv o=rsl1.csv
$ more rsl1.csv
fld
20120304121212
20101204011309
```
Example 2: Add Results to New Column

Store results in \texttt{fld2} from format conversion in \texttt{fld1} field, specify the format by "\texttt{\%Y/\%m/\%d}" in \texttt{c=} parameter. Use \texttt{-A} option to save results in \texttt{f2} field.

```bash
$ more dat2.csv
fld,fld2
2010/11/20,2010/11/21
2010/1/1,2010/1/2
2011/01/01,2010/01/02
2010/1/01,2010/1/02
$ mdformat f=fld:f1,fld2:f2 c=%Y/%m/%d i=dat2.csv -A o=rsl2.csv
$ more rsl2.csv
fld,fld2,f1,f2
2010/11/20,2010/11/21,20101120,20101121
2010/1/1,2010/1/2,20100101,20100102
2011/01/01,2010/01/02,20110101,20100102
2010/1/01,2010/1/02,20100101,20100102
```

Example 3: Case of failed extraction

The date format in \texttt{fld} field is saved as "Year Month Day Time:Minute:Second", "\texttt{\%Y \%m \%d \%H:\%M:\%S}" is specified in \texttt{c=} parameter. However, it failed since the format is different in different rows.

```bash
$ more dat3.csv
fld
2010 11 20 12:34:56
2011 01 01 23:34:56
2010 1 01 123455
$ mdformat f=fld:f1 c='%Y %m %d %H:%M:%S' i=dat3.csv -A o=rsl3.csv
$ more rsl3.csv
fld,f1
2010 11 20 12:34:56,20101120123456
2011 01 01 23:34:56,20110101233456
2010 1 01 123455,
```

Related Command
CHAPTER 3. COMMAND REFERENCE

3.20 mduprec - Duplicate Record

Duplicate each record. Specify the number of rows to duplicate at n=, and duplicate records based on the field specified at f= parameter.

Format

mduprec f=|n= |i= |o= |-nfn |-nfno |-x |--help |--version |

Parameters

f= Duplicate records based on the values in the field name.
   Duplicate the number of rows for each record based on the numeric values in the column.

n= Number of records to duplicate
   Specify the number of rows to duplicate for each record.

Examples

Example 1: Basic Example

Generate multiple records of the data based on the numeric value in the Quantity field. Records containing NULL values will not be duplicated.

```bash
$ more dat1.csv
store,val
A,2
B,
C,5
$ mduprec f=val i=dat1.csv o=rsl1.csv
#END# kgduprec f=val i=dat1.csv o=rsl1.csv
$ more rsl1.csv
store,val
A,2
A,2
B,
C,5
C,5
C,5
C,5
C,5

Example 2: Define the number of rows to duplicate

Duplicate two rows (n=2) for each record in the dataset.

```bash
$ mduprec n=2 i=dat1.csv o=rsl2.csv
#END# kgduprec i=dat1.csv n=2 o=rsl2.csv
$ more rsl2.csv
store,val
A,2
A,2
B,
B,
C,5
C,5
C,5
``` 

Related Commands

mcount : Reverse the operation of mduprec.

mwindow : Copy and shift a specified number of records.
### 3.21 mfldname - Rename Field

Specify the field to rename at `f=` and the new column name at `n=`. Add the `-q` option to revert to the field header names as appeared in Ver. 1, and removes the sort order symbols appended to field names in output for 2.0 commands.

**Format**

```
mfldname f=! n= [-nfni] [i=] [o=} [-nfno] [-x] [-q] [--help] [--version]
```

**Parameters**

- `f=` Specify the field name to change (current field name:new field name).
  - The original field name will not change if this parameter is not set.
- `n=` Specify the new target column name.
  - The number of item names must be the same as the number of columns in the data.
- `-nfni` Field name not present in input data. This option cannot be used with `f=`.

**Examples**

**Example 1: Basic Example**

Change column name from `customer` to `cust` and `october` to `oct`.

```
$ more dat1.csv
customer,itemID,october
a,xx,11
b,yy,122
c,zz,
$ mfldname f=customer:cust,october:oct. i=dat1.csv o=rsl1.csv
#END# kgfldname f=customer:cust,october:oct. i=dat1.csv o=rsl1.csv
$ more rsl1.csv
cust,itemID,oct.
a,xx,11
b,yy,122
c,zz,
```

**Example 2: Rename column**

Change field names to `x,y,z`.

```
$ mfldname n=x,y,z i=dat1.csv o=rsl2.csv
#END# kgfldname i=dat1.csv n=x,y,z o=rsl2.csv
$ more rsl2.csv
x,y,z
a,xx,11
b,yy,122
c,zz,
```

**Example 3: Data without field names**

```
$ more dat2.csv
a,xx,11
b,yy,122
c,zz,
$ mfldname -nfni n=x,y,z i=dat2.csv o=rsl3.csv
#END# kgfldname -nfni i=dat2.csv n=x,y,z o=rsl3.csv
$ more rsl3.csv
x,y,z
a,xx,11
```
Example 4: Remove sort order symbols in field names

```
$ more dat3.csv
customer%r,itemID,october
c,zz.
b,yy,122
a,xx,11
$ mfldname -q i=dat3.csv o=rsl4.csv
#END# kgfldname -q i=dat3.csv o=rsl4.csv
$ more rsl4.csv
customer,itemID,october
c,zz.
b,yy,122
a,xx,11
```

Related Command

**mcut**: Performs similar function to **mfldname**, however, the operation is more complicated to rename certain items. Nevertheless, the operation is slightly faster using **mfldname**.
3.22 mfsort - Sort Field

Sort according to the values of the specified fields at f= within each record (in default ascending order by character string). Note that this does not change the sequence of field names.

Format

mfsort f= [-r] [-n] [i=] [o=] [-nf] [-nfno] [-x] [--help] [--version]

Parameters

f= Specify multiple fields where data items are sorted. The result remains the same when one field is defined.
-n Arrange in numerical order.
-r Arrange in reverse order.

Examples

Example 1: Basic Example

Arrange the values in v1,v2,v3 in ascending order for each record, and output the data items in sequential order corresponding to fields v1,v2,v3.

```
$ more dat1.csv
id,v1,v2,v3
1,b,a,c
2,a,b,a
3,b,e
$ mfsort f=v* i=dat1.csv o=rsl1.csv
#END# kgfsort f=v* i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3
1,a,b,c
2,a,a,b
3,,b,e
```

Example 2: Descending Order

Add -r to arrange in descending order.

```
$ mfsort f=v* -r i=dat1.csv o=rsl2.csv
#END# kgfsort -r f=v* i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3
1,c,b,a
2,b,a,a
3,,b,e
```

Related Command
3.23 mhashavg - Compute Average with Hash Function

Calculate the average of data series specified at \texttt{f=} parameter based on the key at \texttt{k=} with hash function.

The processing speed of this command is faster than \texttt{mavg} since the key fields do not have require prior sorting. However, variation in key lengths (different length of strings in field) will slow down the processing speed.

**Format**

\texttt{mhashavg f=} \texttt{[hs=]} \texttt{[k=]} \texttt{[-n]} \texttt{[i=} \texttt{[o=} \texttt{[-nf]} \texttt{[-nfo] [-x]} \texttt{[precision=} \texttt{[--help] [--version]}}

**Parameters**

\texttt{f=} Calculate the average of the field name (Multiple fields can be specified).

Specify the new field name after colon ".:". Example: \texttt{f=}Quantity:AverageQuantity.

\texttt{k=} Calculate the average on the data series based on the key field(s) (Multiple keys can be specified).

This command do not use aggregate key break processing, prior sorting is not required.

\texttt{hs=} Hash size (Default value: 199999)

Refer to \texttt{mhashsum} for related information.

\texttt{-n} Return NULL in output if there are null values in \texttt{f=}.

**Example**

**Example 1: Basic Example**

Calculate the average \texttt{Quantity} and average \texttt{Amount} for each \texttt{Customer}.

```
$ more dat1.csv
Customer,Quantity,Amount
A,1,
B,,15
A,2,20
B,3,10
B,1,20
$ mhashavg k=Customer f=Quantity,Amount i=dat1.csv o=rsl1.csv
#END# kghashavg f=Quantity,Amount i=dat1.csv k=Customer o=rsl1.csv
$ more rsl1.csv
Customer,Quantity,Amount
A,1.5,20
B,2,15
```

**Example 2: NULL value in output**

The output returns NULL if there NULL value is present in \texttt{Quantity} and \texttt{Amount}. Use \texttt{-n} option to print the null value.

```
$ mhashavg k=Customer f=Quantity,Amount -n i=dat1.csv o=rsl2.csv
#END# kghashavg -n f=Quantity,Amount i=dat1.csv k=Customer o=rsl2.csv
$ more rsl2.csv
Customer,Quantity,Amount
A,1.5,
B,,15
```

**Remarks**

Refer to the benchmark at \texttt{mhashsum} to find out more on processing speed.
3.23. MHASHAVG - COMPUTE AVERAGE WITH HASH FUNCTION

Related commands

mavg : Compute average
mhashsum : Compute hash total value
3.24 mhashsum - Compute Total Sum Using Hash

Calculate the hash total value of the column specified at the \( f = \) parameter for each line item based on the key specified at the \( k = \) parameter.

The processing speed of this command is faster than \texttt{msum} since key fields do not require prior sorting. However, variation in length of keys (different length of strings in field) will slow down the processing speed.

User shall assess the usage of \texttt{mhashsum} and \texttt{msum} based on the contents of the data (Refer to "Benchmark" in the second half of this manual).

**Format**

\[
\texttt{mhashsum} \hspace{1em} f= \hspace{1em} \text{[hs=]} \hspace{1em} \text{[k=} \hspace{1em} \text{]} \hspace{1em} \text{[-n]} \hspace{1em} \text{[i=} \hspace{1em} \text{]} \hspace{1em} \text{[o=} \hspace{1em} \text{]} \hspace{1em} \text{[-nfn]} \hspace{1em} \text{[-nfno]} \hspace{1em} \text{[-x]} \hspace{1em} \text{[precision=]} \hspace{1em} \text{[--help]} \hspace{1em} \text{[--version]}
\]

**Parameters**

- \( f = \) Compute the sum of of values in the column specified (Multiple fields can be specified)
  
  Specify the new field name after colon ":". Example \( f=\text{Quantity:TotalQuantity} \).

- \( k = \) Calculate the sum on the data series based on the key field(s) (Multiple keys can be specified.)
  
  This command do not use \textit{aggregate key break processing}, prior sorting is not required.

- \( hs = \) Hash size [Default value: 199999]
  
  User shall specify the key size for data processing based on speed and memory consumption optimisation requirements. Prime number should be used as hash table size.
  
  The processing speed will slow down if the hash table is not big enough for data with large key size.
  
  A larger hash table will speed up processing but will also require more memory (Refer to "Benchmark" in the second half of this manual).

  Estimating memory requirements: \( K^* (24 + F^*16) \) byte, \( K \): key size, \( F \): number of fields specified \( f = \) parameter.

- \( -n \) Return NULL in output if there are null values in \( f = \).

**Example**

**Example 1: Basic Example**

Calculate the total \texttt{Quantity} and total \texttt{Amount} for each \texttt{Customer} using the hash function.

\[
\texttt{$ more dat1.csv}
\]

\[
\text{Customer,Quantity,Amount}
\]

\[
A,1, \\
B,15 \\
A,2,20 \\
B,3,10 \\
B,1,20
\]

\[
\texttt{mhashsum k=Customer f=Quantity,Amount i=dat1.csv o=rsl1.csv}
\]

\[
\text{#END# kghashsum f=Quantity,Amount i=dat1.csv o=rsl1.csv}
\]

\[
\texttt{$ more rsl1.csv}
\]

\[
\text{Customer,Quantity,Amount}
\]

\[
A,3,20 \\
B,4,45
\]

**Example 2: NULL value in output**

The output returns NULL if NULL value is present in \texttt{Quantity} and \texttt{Amount}. Use \( -n \) option to print the null value.

\[
\texttt{$ mhashsum k=Customer f=Quantity,Amount -n i=dat1.csv o=rsl2.csv}
\]

\[
\text{#END# kghashsum -n f=Quantity,Amount i=dat1.csv o=rsl2.csv}
\]

\[
\texttt{$ more rsl2.csv}
\]

\[
\text{Customer,Quantity,Amount}
\]

\[
A,3,20 \\
B,4,45
\]
Overview of algorithm

The `mhashsum` command uses a hash method known as separate chaining. In this method, a sequence known as a hash table is created, containing all the keys that hash to the same value. The hash function converts and stores the character string containing the keys into integer (hash values) from 0 to M. Two or more keys with the same hash value (conflict keys) will be stored in a linked list from the conflicted slot of hash table. The address of keys are stored in sequential order, but the list is searched in linear order. Thus, the lookup procedure will scan all its entries, and the processing speed decreases with more key conflicts. The default hash size for `mhashsum` is 199999, if the key size is up to 200,000, the average list size is 1 less of the key size. Multiple key conflicts may occur even if the key size is small depending on the data content and structure. The key size can be changed at the `hs=` parameter (maximum value: 1999999).

Benchmark test

Method of Benchmark test

Compare the computation speed of `mhashsum` command (hash size: 199,999) and `msum` command (sort data using `msort` command beforehand) on 13 different types of data with 10 to 1,000,000 key sizes.

Sample data with two columns (key and `fld`) and 500 million rows of random values is generated as shown in the following table. The key is a 6 digit fixed-length numerical value and the `fld` is a 3 digit number.

| Table 3.3: Table 1: Sample data for Benchmark test |
|-------|------|
| key   | fld  |
| 100020| 120  |
| 100007| 107  |
| 100029| 129  |
| 100065| 165  |
| 100030| 130  |
| 100088| 188  |
| 100055| 155  |
| 100093| 193  |
| 100072| 172  |

Commands for benchmark

Using the `mhashsum` method

```
$ time mhashsum k=key f=fld i=dat.csv o=/dev/null
```

Using the `msort+msum` method

```
$ time msort i=dat.csv msum k=key f=fld o=/dev/null
```

Experiment Results

The proceeding speed of `mhashsum` is five times faster than sorting when the key size is small (10,000). As the key size increases, the difference between the two methods is reduced, the processing speed of the two methods are the same when the key size is more than 800,000.

The following is a guideline on the usage of `msum` or `mhashsum`, actual results varies depending on the distribution of the key values.
**Table 3.4: Table 2: Comparison of processing speed for mhashsum and msum(msort+msum)**

<table>
<thead>
<tr>
<th>key size</th>
<th>mhashsum(a)(second)</th>
<th>msort+msum(b)(second)</th>
<th>ratio(a/b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.672</td>
<td>2.955</td>
<td>0.227</td>
</tr>
<tr>
<td>1,000</td>
<td>0.731</td>
<td>3.981</td>
<td>0.184</td>
</tr>
<tr>
<td>10,000</td>
<td>0.814</td>
<td>4.201</td>
<td>0.194</td>
</tr>
<tr>
<td>100,000</td>
<td>1.793</td>
<td>4.291</td>
<td>0.418</td>
</tr>
<tr>
<td>200,000</td>
<td>2.241</td>
<td>4.336</td>
<td>0.517</td>
</tr>
<tr>
<td>300,000</td>
<td>2.604</td>
<td>4.394</td>
<td>0.593</td>
</tr>
<tr>
<td>400,000</td>
<td>2.993</td>
<td>4.448</td>
<td>0.673</td>
</tr>
<tr>
<td>500,000</td>
<td>3.380</td>
<td>4.497</td>
<td>0.752</td>
</tr>
<tr>
<td>600,000</td>
<td>3.793</td>
<td>4.579</td>
<td>0.828</td>
</tr>
<tr>
<td>700,000</td>
<td>4.128</td>
<td>4.618</td>
<td>0.894</td>
</tr>
<tr>
<td>800,000</td>
<td>4.514</td>
<td>4.667</td>
<td>0.967</td>
</tr>
<tr>
<td>900,000</td>
<td>4.901</td>
<td>4.707</td>
<td>1.041</td>
</tr>
<tr>
<td>1,000,000</td>
<td>5.352</td>
<td>4.771</td>
<td>1.122</td>
</tr>
</tbody>
</table>

**Benchmark environment**

iMac, Mac OS X 10.5 Leopard, 2.8GHz Intel Core 2 Duo, 4GB memory

**Related commands**

- **msum**: Same computation function but requires key break process.
- **mhashavg**: Compute average using the same hash function
3.25 mkeybreak Keybreak Point

Add an indicator to first rows and last rows of each key specified at k= parameter. The first row is indicated as 1 in the top field, and the same indicator is added to last row in the bot column. Records that are not in the first row nor last row will appear as NULL values.

Format

mkeybreak k= [s=] [a=] [i=] [o=] [-nfn] [-nfno] [-x] [-q] [--help] [--version]

Parameter

k= Specify field names (multiple fields can be specified) of the aggregate key.
s= After sorted by specified fields (multiple fields can be specified), the first and last rows are marked.
a= Specify the field name for top and bottom indicators in the output. [default value: top,bot]

Examples

Example 1: Basic Example

Sort the records by ˇverb—k1— field, add an indicator (1) to the first record (top field) and last record (bottom field) where k1 is key field.

```
$ more dat1.csv
id,k1,k2,val
1,A,a,1
2,A,b,2
3,A,b,3
4,B,a,4
5,B,a,5
$ mkeybreak k=k1 i=dat1.csv o=rsl1.csv
#END# kgkeybreak i=dat1.csv k=k1 o=rsl1.csv
$ more rsl1.csv
id,k1%0,k2,val,top,bot
1,A,a,1,1,
2,A,b,2,,
3,A,b,3,,
4,B,a,4,1,
5,B,a,5,,1
```

Example 2: 2 key fields

After fields k1 and k2 are sorted, the beginning of key field k1 (topfield) and end (bottomfield) is marked (1).

```
$ mkeybreak s=k1,k2 k=k1 i=dat1.csv o=rs12.csv
#END# kgkeybreak i=dat1.csv k=k1 o=rs12.csv s=k1,k2
$ more rs12.csv
id,k1,k2,val,top,bot
1,A,a,1,1,
2,A,b,2,,
3,A,b,3,,
4,B,a,4,1,
5,B,a,5,,1
```

Related Command
CHAPTER 3. COMMAND REFERENCE

3.26  mjoin - Join field(s) from Reference File

Compare the key field(s) from the input file specified at the k= parameter with the ones from the reference file, field from the reference file specified at f= parameter are joined for records with common key values in both files. The key fields from the reference file must be unique. Use the mnjoin command when there are more than one record with the same key values in the reference file. If f= is not set, all columns are joined except the key field in reference file.

Format


Parameters

k= Specify the key field name from input data field and at the K= parameter.

Join records with the same field(s) from reference data.

Set to NULL value when key fields do not match with the fields from reference file specified at K=.

f= Field name of the reference file to join.

When this parameter is not set, all fields except the key field will be joined.

m= Specify list of reference file(s).

Read from standard input when this parameter is not set (used when input data is specified at i=).

K= List of field names from reference data for matching.

Specify key field(s) from reference data to join with records with the key field from the input data defined at the k= parameter.

Set to NULL value when key fields do not match with the fields from input file specified at k=.

The key field(s) name from the reference file does not need to be the same as the k= parameter.

-n Output NULL values when reference data does not consist of input data.

-N Output NULL values when input data does not consist of reference data.

Examples

Example 1: Basic Example

Join the field cost from the reference file for records where the values of the item column from the input file is the same as the values in item column in the reference file.

```
$ more dat1.csv
item, date, price
A, 20081201, 100
A, 20081213, 98
B, 20081002, 400
B, 20081209, 450
C, 20081201, 100
$ more ref1.csv
item, cost
A, 50
B, 300
E, 200
$ mjoin k=item f=cost m=ref1.csv i=dat1.csv o=rsl1.csv
END# kgjoin f=cost i=dat1.csv k=item m=ref1.csv o=rsl1.csv
$ more rsl1.csv
itemX0, date, price, cost
A, 20081201, 100, 50
A, 20081213, 98, 50
B, 20081002, 400, 300
B, 20081209, 450, 300
```
Example 2: Output unmatched data

Join the cost field for records with common key values in the item field from the input file and reference file, join cost item. At the same time, join all keys from the reference file if the value in the reference file is not in input data range, and set as NULL values.

```
$ mjoin k=item f=cost m=ref1.csv -n -N i=dat1.csv o=rsl2.csv
#END# kgjoin -N -n f=cost i=dat1.csv k=item m=ref1.csv o=rsl2.csv
$ more rsl2.csv
item%0,date,price,cost
A,20081201,100,50
A,20081213,98,50
B,20081002,400,300
B,20081209,450,300
C,20081201,100,
E,,,200
```

Related Command

- mnjoin: Use mnjoin to duplicate key from reference file.
- mpaste: Join according to row number.
- mcommon: Use mcommon to select records instead of joining.
### 3.27 mmbucket - Multi-dimensional Uniform Bucket Partition

Segment numerical values in multiple fields specified at $f=$ into equal sized buckets. For example, set $f=a,b,c$ and $n=5$, mbucket command segments fields $a,b,c$ into 5 equal buckets. Whereas mmbucket allocate items $a,b,c$ into 3 dimensional space for each bucket (bucket number becomes $5^3 = 125$) such that each interval is distributed as evenly as possible.

**Format**

\[
\]

**Parameters**

- $f=$ Values in this field(s) (multiple fields can be specified) is(are) partitioned. When there are multiple fields, the number of dimensions is determined based on equal number of buckets. When 1 field is specified, the result is the same as using mbucket.
- $-x,-nfn$ options can be used to specify field number (0~n).
- $n=$ Each bucket size corresponds to a field specified at $f=$. The number of items defined here is the same as the number of fields specified at $f=$. However, if there is only 1 number specified, the same partition number is applied to other field.
- $F=$ Output format [default value: 1]
  - 0: Display bucket number.
  - 1: Display value range of buckets.
  - 2: Display both bucket number and value range.
- $k=$ Unique key field(s) (multiple fields can be specified) to retrieve rows of data for partitioning into buckets.
- $O=$ Specifies the output file with numeric range of each bucket from the items specified in the $f=$ parameter.
- $-ms$ When partitioning buckets for each field sequentially, the first item is changed multiple times during the bucket partition trial. For further information, refer to the "Introduction to Algorithms" to find out the best possible solution.
- $-r$ Display bucket number in reverse order.

**Examples**

**Example 1: Basic Example**

Partition the number of records in column $x,y$ into two multi-dimensional equal subsets. At the same time, save the numeric range of each bucket in the file named rng.csv.

```bash
$ more dat1.csv
id,x,y
A,2,7
B,6,7
C,5,6
D,7,5
E,6,4
F,1,3
G,3,3
H,4,2
I,7,2
J,1,1
$ mmbucket f=x:xb,y:yb n=2,2 O=rng.csv i=dat1.csv o=rsl1.csv
calculating on dimension ... #0 #1 done. VAR=30 updated!
calculating on dimension ... #0 #1 done. VAR=28 updated!
#END# kgmbucket O=rng.csv f=x:xb,y:yb i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,x,y,xb,yb
A,2,7,1,2
B,6,7,2,2
C,5,6,2,2
D,7,5,2,2
```

$ more dat1.csv
id,x,y
A,2,7
B,6,7
C,5,6
D,7,5
E,6,4
F,1,3
G,3,3
H,4,2
I,7,2
J,1,1
$ mmbucket f=x:xb,y:yb n=2,2 O=rng.csv i=dat1.csv o=rsl1.csv
calculating on dimension ... #0 #1 done. VAR=30 updated!
calculating on dimension ... #0 #1 done. VAR=28 updated!
#END# kgmbucket O=rng.csv f=x:xb,y:yb i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,x,y,xb,yb
A,2,7,1,2
B,6,7,2,2
C,5,6,2,2
D,7,5,2,2
```
Example 2: Output Format

Partition the number of records in column x,y into two multi-dimensional equal subsets based on the id field. The output format shall display both the bucket number and numeric range.

Overview of algorithm

Assuming data set D consists of two columns A, B with numerical values. The numeric range in column A and B is divided into K bits and L bits respectively. mmbucket determines how to divide the data equally into segments (buckets). Dispersion is used as a basis to measure the uniformity of distribution. The variance value minimizes the number of buckets partitions based on the heuristics described below, which is not guaranteed to be optimal. The algorithm is described as follows.

1. Partition data in field A by one dimensional uniform partition.

2. Determine the partition of A from step 1, partition the range of numerical values in B into buckets. The evaluation criteria for two-dimensional bucket partition is based on the partitions. Dynamic programming and one-dimensional bucket partition is used to divide buckets into equal sized partitions.

---

1This algorithm is developed by Professor Naoki Kato (Kyoto University, Graduate School of Engineering).
3. Next, using the partitions of $B$ obtained from step 2, partition the range of numerical values in $A$. As in step 2, the values of two dimensional bucket partition is used as an evaluation criteria.

4. Repeat step 2 and 3 recursively until improvement of variance is minute.

5. Return partition output.

During implementation, when $A$ and $B$ is parts are switched in step 1, the solution returns the one with more optimal results. The same algorithm applies to uniform buckets partition for three or more dimensions.

**Comparison of mbucket and mmbucket**

The following explains how multi-dimensional bucket partition operates compared to one-dimensional bucket partition. Table 3.5 shows two columns $x$ and $y$, with 10 rows data of id from A to J.

Using this data, each of $x$ and $y$ is divided into two partitions, thus create a total of four buckets as shown in Figure 1. The results of one-dimensional bucket partition on $x$ and $y$ is shown in column $x_1$, $y_1$. In addition, the results of multi-dimensional bucket partition is shown in column $x_2$ and $y_2$ as displayed in Figure 2.

Variance (sum of squares of each bucket : for more information, refer to formulation of mbucket) of one dimensional bucket partition is computed as $Var_a = 1^2 + 4^2 + 4^2 + 1^2 = 34$, and $Var_b = 1^2 + 3^2 + 3^2 + 3^2 = 28$ for two-dimensional bucket partition. During one-dimensional bucket partition $x$ and $y$ is treated independently to optimize solution with minimal variance (i.e. each is divided into 5). The solution of one dimensional bucket partition is inferior to two-dimensional bucket partition. One dimensional bucket partition can be improved by using the multi-dimensional bucket partition. However, multi-dimensional partition requires a lot of CPU time dependent on the contents of the data.

<table>
<thead>
<tr>
<th>Data</th>
<th>mbucket</th>
<th>mmbucket</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>$x$</td>
<td>$y$</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>H</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>I</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>J</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 3.5:** Sample result of output from mbucket, mmbucket

![Figure 3.1: One-dimensional partition (mbucket)](image1)

![Figure 3.2: Two-dimensional partition (mmbucket)](image2)

**Precision experiment using on variety of fields**

The following compares the precision of mmbucket and mbucket using real data collected by Tropical Atmosphere Ocean (TAO) Project. The project is designed for the study of year-to-year climate variations related to El Nino
the Southern Oscillation (ENSO), and provides valuable in-situ data collection of high quality oceanographic
and surface meteorological data for monitoring, forecasting, and understanding of climate swings associated
with El Nino and La Nina. Snapshot of the data is shown in the following table.

### Table 3.6: Data set used for precision experiment

<table>
<thead>
<tr>
<th>id</th>
<th>year</th>
<th>month</th>
<th>day</th>
<th>date</th>
<th>latitude</th>
<th>longitude</th>
<th>zonewinds</th>
<th>merwinds</th>
<th>humidity</th>
<th>air_temp</th>
<th>sstemp</th>
</tr>
</thead>
<tbody>
<tr>
<td>4060</td>
<td>93</td>
<td>5</td>
<td>9</td>
<td>930509</td>
<td>-0.02</td>
<td>-109.96</td>
<td>-2.1</td>
<td>2.1</td>
<td>81.2</td>
<td>26.8</td>
<td>27.02</td>
</tr>
<tr>
<td>4061</td>
<td>93</td>
<td>5</td>
<td>10</td>
<td>930510</td>
<td>-0.02</td>
<td>-109.96</td>
<td>-3.4</td>
<td>1.4</td>
<td>84.2</td>
<td>26.95</td>
<td>26.91</td>
</tr>
<tr>
<td>4062</td>
<td>93</td>
<td>5</td>
<td>11</td>
<td>930511</td>
<td>-0.02</td>
<td>-109.96</td>
<td>-3.8</td>
<td>2.2</td>
<td>84.9</td>
<td>26.98</td>
<td>26.78</td>
</tr>
<tr>
<td>4063</td>
<td>93</td>
<td>5</td>
<td>12</td>
<td>930512</td>
<td>-0.02</td>
<td>-109.96</td>
<td>-3</td>
<td>1.5</td>
<td>86.9</td>
<td>26.93</td>
<td>26.74</td>
</tr>
<tr>
<td>4064</td>
<td>93</td>
<td>5</td>
<td>13</td>
<td>930513</td>
<td>-0.02</td>
<td>-109.96</td>
<td>-4.5</td>
<td>1.9</td>
<td>87.6</td>
<td>27.01</td>
<td>26.82</td>
</tr>
<tr>
<td>4065</td>
<td>93</td>
<td>5</td>
<td>14</td>
<td>930514</td>
<td>-0.02</td>
<td>-109.96</td>
<td>-5</td>
<td>1.3</td>
<td>85.6</td>
<td>26.96</td>
<td>26.68</td>
</tr>
</tbody>
</table>

The precision experiment uses attributes with numeric values (7 attributes from latitude to sea surface temperature) for bucket partition. The number of record rows is 93,935. Statistics for each attribute is shown in Table 3. "Types of data values" significantly affects computation time of bucket partition.

### Table 3.7: Various statistics of numerical data

<table>
<thead>
<tr>
<th>Type</th>
<th>Measurement</th>
<th>Minimum</th>
<th>Mean</th>
<th>Maximum</th>
<th>Type</th>
<th>Measurement</th>
<th>Minimum</th>
<th>Mean</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Latitude</td>
<td>-70.8</td>
<td>42.7</td>
<td>170.0</td>
<td>Latitude</td>
<td>-80.1</td>
<td>35.7</td>
<td>120.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Longitude</td>
<td>128.7</td>
<td>3.42</td>
<td>1.3</td>
<td>Longitude</td>
<td>128.7</td>
<td>3.42</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>North-south wind speed</td>
<td>-10.7</td>
<td>-10.6</td>
<td>-0.1</td>
<td>North-south wind speed</td>
<td>-10.7</td>
<td>-10.6</td>
<td>-0.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Zonal wind speed</td>
<td>14.3</td>
<td>13</td>
<td>99.9</td>
<td>Zonal wind speed</td>
<td>14.3</td>
<td>13</td>
<td>99.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sea surface temperature</td>
<td>228</td>
<td>206</td>
<td>385</td>
<td>Sea surface temperature</td>
<td>228</td>
<td>206</td>
<td>385</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Humidity</td>
<td>-0.02</td>
<td>81.3</td>
<td>93</td>
<td>Humidity</td>
<td>-0.02</td>
<td>81.3</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Temp</td>
<td>385</td>
<td>1104</td>
<td>1201</td>
<td>Temp</td>
<td>385</td>
<td>1104</td>
<td>1201</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sea surface temp</td>
<td>206</td>
<td>385</td>
<td>1104</td>
<td>Sea surface temp</td>
<td>206</td>
<td>385</td>
<td>1104</td>
<td></td>
</tr>
</tbody>
</table>

The comparison of variance for one-dimensional bucket partition (mbucket) and two-dimensional bucket partition (mmbucket) based on a total of 21 combinations from the 7 attributes are distributed into two columns as shown in Table 3.8.

For example, the first row shows the comparison results of bucket partition on latitude (mbucket) with El Nino and La Nina. The number of record rows is 93,935. Statistics for each attribute is shown in Table 3. "Types of data values" significantly affects computation time of bucket partition.

### Table 3.8: Comparison of two-dimensional partition accuracy of mbucket and mmbucket

<table>
<thead>
<tr>
<th>Item</th>
<th>Item 2</th>
<th>Variance (mbucket / mmbucket)</th>
<th>Comparison of variance (mbucket/mmbucket)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>5 x 5 partition</td>
<td>10 x 10 partition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 x 15 partition</td>
<td>20 x 20 partition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mbucket</td>
<td>mmbucket</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 x 5</td>
<td>10 x 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 x 15</td>
<td>20 x 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.996</td>
<td>0.987</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.992</td>
<td>0.988</td>
</tr>
</tbody>
</table>
CHAPTER 3. COMMAND REFERENCE

Speed Comparison

Next, the experiments are carried out to compare the differences in speed by against different number of partitions. The execution time is computed using the same data \( \text{temperature} \) \( \text{sea surface temperature} \) \( \text{latitude} \) \( \text{longitude} \) with 5 to 40 partitions at increments of 5. The results are shown in Table 3.9. The execution time of \texttt{mbucket} is consistent regardless of the number of partitions. However, more time is required to compute the number of partitions for two dimensional partitions. This is due to the fact that the algorithm is not efficient in selecting multidimensional orthogonal numeric range required by computation. The partition speed will be noted as possible improvements in the next version.

<table>
<thead>
<tr>
<th>Number of buckets</th>
<th>Temperature ( \text{mbucket} )</th>
<th>Sea Surface Temperature ( \text{mmbucket} )</th>
<th>Latitude ( \text{mbucket} )</th>
<th>Longitude ( \text{mmbucket} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.221</td>
<td>2.21</td>
<td>0.216</td>
<td>0.67</td>
</tr>
<tr>
<td>10</td>
<td>0.227</td>
<td>3.90</td>
<td>0.216</td>
<td>1.67</td>
</tr>
<tr>
<td>15</td>
<td>0.233</td>
<td>10.4</td>
<td>0.230</td>
<td>3.28</td>
</tr>
<tr>
<td>20</td>
<td>0.231</td>
<td>26.1</td>
<td>0.228</td>
<td>7.13</td>
</tr>
<tr>
<td>25</td>
<td>0.232</td>
<td>32.9</td>
<td>0.237</td>
<td>13.3</td>
</tr>
<tr>
<td>30</td>
<td>0.236</td>
<td>46.7</td>
<td>0.236</td>
<td>11.4</td>
</tr>
<tr>
<td>35</td>
<td>0.237</td>
<td>62.3</td>
<td>0.240</td>
<td>15.2</td>
</tr>
<tr>
<td>40</td>
<td>0.237</td>
<td>80.1</td>
<td>0.237</td>
<td>25.8</td>
</tr>
</tbody>
</table>

Related Command

\texttt{mbucket} : This command processes one-dimensional bucket partition for each field even when more than one field is specified.
3.28 mmvavg - Calculate Moving Average

Calculate the moving average. The three different ways to calculate moving average include simple moving average (SMA), weighted moving average (WMA), and exponential moving average (EMA).

The value of \( t \) time is expressed by \( x_t \), and period is represented by \( m \) as defined in several formulas of moving average (3.1,3.2,3.3).

\[
SMA_t = \frac{1}{m} \sum_{i=0}^{m-1} x_{t-i}
\]

(3.1)

\[
WMA_t = \sum_{i=0}^{m-1} \frac{m-i}{S} x_{t-i}, \quad S = \sum_{i=1}^{m} i
\]

(3.2)

\[
EMA_t = \alpha x_t + (1 - \alpha) EMA_{t-1}
\]

(3.3)

Format

mmvavg [s=] [k=] f= [t=] [-exp|-w] [alpha=} [skip=] [i=] [o=] [-nfn] [-nfno] [-x] [-q] [precision=} [--help] [--version]

Parameters

s=  After the specified field is sorted (multiple fields can be specified), moving average is calculated.
   s= parameter is required when -q option is not specified.

k=  Aggregate records using the specified field name(s) (multiple fields can be used as unit of calculation).

f=  Compute the moving average of the field(s) (multiple fields can be specified).

\( t= \) Interval numbers of integers greater than 1.
   When -exp is used with alpha=, the \( t= \) parameter do not need to be defined.

-w  Linear weighted moving average.

-exp  Exponential smoothing moving average.

alpha=  Use a real number as smoothing coefficient when -exp is specified.
   The default value of alpha is \( \text{alpha=2/(value of t+1)} \).

skip=  Specify the number of rows to hide from the top in the output.
   Default value: \( \text{skip=(value of t= -1)} \), \text{skip=0 when -exp is specified}.

Examples

Example 1: Basic Example

The first row is not printed as there is less than the number of required intervals for computation.

```bash
$ more dat1.csv
id,value
1,5
2,1
3,3
4,4
5,4
6,6
7,1
8,4
9,7

$ mmvavg s=id f=value t=2 i=dat1.csv o=rsl1.csv
```
Example 2: Basic Example 2

The first row is not printed as there is less than the number of required intervals for computation.

```
$ mmvavg s=id f=value t=2 -w i=dat1.csv o=rsl2.csv
#END# kgmvavg -w f=value i=dat1.csv o=rsl2.csv s=id t=2
$ more rsl2.csv
id%0,value
2,2.333333333
3,2.333333333
4,3.666666667
5,5
6,5.333333333
7,7.666666667
8,3
9,5
```

Example 3: Basic Example 3

Exponential smoothing moving average (-exp) includes the first row in the output.

```
$ mmvavg s=id f=value t=2 -exp i=dat1.csv o=rsl3.csv
#END# kgmvavg -exp f=value i=dat1.csv o=rsl3.csv s=id t=2
$ more rsl3.csv
id%0,value
1,5
2,2.333333333
3,2.777777778
4,3.592592593
5,3.864197531
6,3.288065844
7,2.429355281
8,3.47645176
9,5.82548392
```

Example 4: An example of assigning key

```
$ more dat2.csv
id,key,value
1,a,5
2,a,1
3,a,3
4,a,4
5,a,4
6,b,6
7,b,1
8,b,4
9,b,7
$ mmvavg s=key id=k key f=value t=2 i=dat2.csv o=rsl4.csv
#END# kgmvavg f=value i=dat2.csv k=key o=rsl4.csv s=key t=2
$ more rsl4.csv
id,key,value
2,a,3
3,a,2
4,a,3.5
5,a,4
7,b,3.5
```
Example 5: Display all records including those that are less than the defined intervals

```bash
$ more dat3.csv
key,value
a,1
a,2
a,3
a,4
a,5
b,6
b,1
b,4
b,7
$ mmvavg -q k=key f=value t=2 skip=0 i=dat3.csv o=rsl5.csv
END# kgmvavg -q f=value i=dat3.csv k=key o=rsl5.csv skip=0 t=2
$ more rsl5.csv
key,value
a,1
a,1.5
a,2.5
a,3.5
a,4.5
b,6
b,3.5
b,2.5
b,5.5
```

Related Commands

- `mmvstats`: Specify the average as well as various types of statistics.
- `mmvsim`: Compute bivariate statistics.
- `mwindow`: Computes statistics on sliding window data which cannot be computed using `mmvstats`. 
CHAPTER 3. COMMAND REFERENCE

3.29 mmvsim - Sliding Windows Similarity Measure

Compute the similarity measure (bivariate statistics) using settings of sliding window. This is the sliding windows version of the msim command. The main difference between the two is that msim can only carry out similarity calculation for 1 target object, whereas mmvsim quantifies similarity between two objects.

Format

```
mmvsim [s=] [k=] f= c= [t=] [skip=] [i=] [o=] [-nfn] [-nfo] [-x] [-q] [precision=] [--help] [--version]
```

Parameters

- `s=`: After the specified field is sorted (multiple fields can be specified), a variety of similarity computation is carried out. The `s=` parameter is required if `-q` is not specified.
- `k=`: Aggregate records using the specified field name(s) (multiple fields can be specified).
- `f=`: Field name(s) (multiple fields can be specified) for computation.
- `t=`: Integer intervals that is greater than 1.
- `c=`: Define measures of similarity (one from the following).
  - `covar|ucovar|pearson|spearman|kendall|euclid|
  - `cosine|cityblock|hamming|chi|phi|jaccard|support|lift`
  Refer to msim command for detailed definition.
- `skip=`: Specify the number of rows to hide from the top in the output [default value: `skip=(value of t= -1)`]

Example

Example 1: Basic Example

Calculate the Pearson product-moment correlation coefficient for 3 window intervals for fields `x`, `y`.

```
$ more dat1.csv
t,x,y
1,14,0.17
2,11,0.2
3,32,0.15
4,13,0.33
5,8,0.1
6,19,0.56
$ mmvsim s=t t=3 c=pearson f=x,y a=sim i=dat1.csv o=rsl1.csv
$ more rsl1.csv
t%0,x,y,sim
3,32,0.15,-0.8746392857
4,13,0.33,-0.6515529194
5,8,0.1,-0.1164257338
6,19,0.56,0.9986254289
```

Related Commands

- `msim`: Find out the degree of similarity without setting sliding window.
- `mwindow`: Create sliding window data for use with `mmvstats` for the computation of statistics.
- `mmvavg`: This command only computes moving average.
3.30 mmvstats - Compute Statistics of Sliding Window

Calculate various statistics (1 variable) for sliding windows. This a variant of the mstats command with functionality to compute sliding windows.

Format

```
mmvstats [s=] [k=] f= [t=] c= [skip=] [i=} [o=} [-nfn] [-nfno] [-x] [-q] [precision=] [--help] [--version]
```

Parameters

- **s=**: After the specified field is sorted (multiple fields can be specified), various statistics is computed.
- **k=**: Aggregate records using the specified field name(s) (multiple fields can be specified).
- **f=**: Field name(s) (multiple fields can be specified) for computation.
- **t=**: Integer interval greater than 1.
- **c=**: Type of statistics (specify one of the following list)
  - sum|mean|devsq|var|uvar|sd|usd|cv|min
  - max|range|skew|uskew|kurt|ukurt
  - Refer to mstats for detailed definitions.
- **skip=**: Specify the number of rows to hide from the top in the output.

Example

**Example 1: Basic Example**

Calculate sum of sliding window. The first row is not printed as there is less than the required number of intervals for computation.

```
$ more dat1.csv
id,value
1,5
2,1
3,3
4,4
5,4
6,6
7,1
8,4
9,7
$ mmvstats s=id f=value t=2 c=sum i=dat1.csv o=rsl1.csv
#END# kgmvstats c=sum f=value i=dat1.csv o=rsl1.csv s=id t=2
$ more rsl1.csv
id%0,value
2,6
3,4
4,7
5,8
6,10
7,7
8,5
9,11
```

Related Commands

- **mmnavg**: Calculate moving average.
- **mwindow**: Create sliding window data for use with mmvstats for the computation of statistics.
- **mmvsim**: Compute the similarity measure (bivariate statistics).
3.31 mnewnumber - Generate List of Sequential Numbers

Define the start value of the alphabetic sequence at the S= parameter, set the the interval of alphabet sequence at I= parameter, and define the column name of the sequence at a= parameter. The alphabet sequence uses 26 alphabetic characters in base-26 from A to Z (A,B,...,Z,AA,AB,...,AZ,BA,BB,...,ZZ,AAA,AAB,...).

Format

mnewnumber a= [I=] [S=} [l=] [o=] [-nfn] [-nfo] [--help] [--version]

Parameters

a= Specify the field name of the list of new serials. This parameter is not required when -nfn,-nfo option is specified.
I= Interval between the sequence of numbers [default value: 1]
S= Starting value/alphabet(upper case letters) [default value:1] Assign either alphabet or numbers as the starting value of the sequence. A list of serial numbers is generated when the starting numeric value is specified. An alphabet sequence is generated when the starting alphabet is specified (cannot be specified in lowercase).
l= Number of rows to generate [default value:10]

Examples

Example 1: Basic Example

Generate a dataset with 5 sequential numbers starting from 1 incremented by 1. Name the sequence as No..

```bash
$ mnewnumber a=No. I=1 S=1 l=5 o=rsl1.csv
#END# kgNewnumber I=1 S=1 a=No. l=5 o=rsl1.csv
$ more rsl1.csv
No.
1
2
3
4
5
```

Example 2: Change the starting number and interval

Generate a dataset consisting of 5 sequential numbers starting from 10 with an incremental interval of 5. Name the sequence as No..

```bash
$ mnewnumber a=No. I=5 S=10 l=5 o=rsl2.csv
#END# kgNewnumber I=5 S=10 a=No. l=5 o=rsl2.csv
$ more rsl2.csv
No.
10
15
20
25
30
```

Example 3: Generate series of alphabet

Generate a dataset consisting of 5 alphabet sequence starting from A with 1 alphabet in between. Name the sequence as No..

```bash
$ mnewnumber a=No. I=1 S=A l=1 o=rsl3.csv
#END# kgNewnumber I=1 S=A a=No. l=1 o=rsl3.csv
$ more rsl3.csv
No.
A
B
C
D
E
```
Example 4: Generate data without header

Generate a dataset consisting of 11 alphabet sequence starting from B with 3 alphabets in between. Exclude the header from the output.

```
$ mnewnumber -nfn I=3 l=11 S=B o=rsl4.csv
#END# kgNewnumber -nfn I=3 S=B l=11 o=rsl4.csv
$ more rsl4.csv
B  
E  
H  
K  
N  
Q  
T  
W  
Z  
AC 
AF 
```

Related Commands

- **mnewrand**: Generate a dataset with random numbers.
- **mnewstr**: Generate fixed character strings.
### 3.32 mnewrand - Generate Dataset with Random Numbers

Generate real random numbers from the range of 0.0 to 1.0. Use the `-int` option to generate randomized sequences of integers.

This command uses Mersenne Twister to generate random numbers. (Webpage of author, boost library)

#### Format

```
mnewrand a= [max=] [min=] [S=] [l=} [-int] [o=] [-nfn] [-nfno] [--help] [--version]
```

#### Parameters

- **a=** Name of column in the new dataset created. This parameter is not required when `-nfn`, `-nfno` options are specified.
- **max=** Maximum value of random number [default value: INT_MAX] -`int` must be specified with this parameter.
- **min=** Minimum value of random number [default value: 0] -`int` must be specified with this parameter.
- **S=** Random seed [default value: current time]
- **l=** Number of lines [default value: 10] Generate a new dataset with random numbers according to the number of rows specified.
- **-int** Generate random integers.

#### Examples

**Example 1: Basic Example**

Generate 10 rows of random integers. Use a fixed random seed so that it will always return the same sequence of random numbers.

```
$ mnewrand a=rand S=1 o=rsl1.csv
END# kgnewrand S=1 a=rand o=rsl1.csv
$ more rsl1.csv
rand
0.4170219984
0.9971848081
0.7203244893
0.9325573612
0.0001143810805
0.1281244478
0.3023325677
0.9990405154
0.1467558926
0.2360889763
```

**Example 2: Random Integers**

Use random seed 1 to generate 5 rows of random integers with minimum value of 10 and maximum value of 100.

```
$ mnewrand a=rand -int max=1000 min=0 l=5 S=1 o=rsl2.csv
END# kgnewrand -int S=1 a=rand l=5 max=1000 min=0 o=rsl2.csv
$ more rsl2.csv
rand
417
998
721
933
0
```
Example 3: Generate Output without Header

Specify `-nfn` option to generate random number data without header.

```bash
$ mnewrand -nfn l=5 S=1 o=rsl3.csv
#END# kgnewrand -nfn S=1 l=5 o=rsl3.csv
$ more rsl3.csv
0.4170219984
0.9971848081
0.7203244893
0.9325573612
0.0001143810805
```

Related Commands

- **mnnewnumber**: Generate list of sequential numbers.
- **mnnewstr**: Generate a list of character strings.
3.33 **mnewstr - Generate Fixed String Data**

Define the character string to generate at \( v = \) parameter, and pass the new column name at \( a = \) parameter. Multiple columns can be generated at a time.

**Format**

\[
mnewstr \ a= [v=] \ [l=] [o=] [-nfn] [-nfn0] [-help] [-version]
\]

**Parameters**

- **a** = Field name(s) of the new data. Define multiple field names separated with a comma in between the field names. This argument is not required when \(-nfn, -nfn0\) options are specified.
- **v** = Specify the new character string to generate. Define multiple field separated with a comma in between the values. The number of fields must be the same as the number of field names defined at \( a = \).
- **l** = Number of rows of random data to generate [default value: 10].

**Example**

**Example 1: Basic Example**

Generate a new dataset with characters strings `custNo` and `A0001` printed in 5 rows, and name the fields as `attribute` and `code` respectively.

```
$ mnewstr a=attribute,code v=custNo,A0001 l=5 o=rsl1.csv
#END# kgnewstr a=attribute,code l=5 o=rsl1.csv v=custNo,A0001
$ more rsl1.csv
attribute,code
custNo,A0001
custNo,A0001
custNo,A0001
custNo,A0001
custNo,A0001
```

**Related Commands**

- **mnewnumber** : Generate list of sequential numbers in a new dataset.
- **mnewrand** : Generate random numbers in a new dataset.
3.34 mnjoin - Natural Join with Reference File

A natural join selects rows from input data and reference file that have equal values in columns defined at k= parameter, the reference file is specified at the m= parameter, fields in the reference file specified at f= parameter is added through natural join. The difference with mjoin command is that key field(s) in the reference field is not unique.

Given that the input data has \( n \) records with a certain key value, and the reference file has \( m \) records with the same key value, \( n \times m \) records will be generated. In addition, if \( f= \) is not defined, all fields except the key field will be added.

### Format

```
```

### Parameters

- **k=** key field name(s) from the input data for matching
  
  This key field is specified in the input data and at the K= parameter.
  
  Join rows when this fields matches with the fields from the reference data.

- **f=** Specify the field name(s) to join from the reference file.
  
  When this parameter is not defined, all fields except the key field will be joined.

- **m=** Reference file name.
  
  Read from standard input if this parameter is not set. (when i= is specified)

- **K=** Key field name(s) from the reference data for matching
  
  This key field(s) from reference data is compared with the key field(s) from input data specified at k= parameter, fields where records with common key are joined.

  This parameter is not required if the field name in reference file is the same as the one defined at the k=parameter.

- **\-n** Output NULL values when reference data does not consist of input data.

- **\-N** Output NULL values when input data does not consist of reference data.

### Examples

#### Example 1: Basic Example

The **item** field in the input file is compared with the **item** field from the reference file, add **cost** field for records with the same value. There are two records where **item=A** in both input file and reference file, therefore, \( 2 \times 2 = 4 \) rows of **item=A** is written to the output file.

```bash
$ more dat1.csv
item, date, price
A, 20081201, 100
A, 20081213, 98
B, 20081002, 400
B, 20081209, 450
C, 20081201, 100

$ more ref1.csv
item, cost
A, 50
A, 70
B, 300
E, 200

$ mnjoin k=item f=cost m=ref1.csv i=dat1.csv o=rsl1.csv
END# kgnjoin f=cost i=dat1.csv k=item m=ref1.csv o=rsl1.csv

$ more rsl1.csv
item%0, date, price, cost
A, 20081201, 100, 50
A, 20081201, 100, 70
A, 20081213, 98, 50
A, 20081213, 98, 70
B, 20081002, 400, 300
B, 20081209, 450, 300
```
Example 2: Output unmatched data

Use -n to print records in the input data that do not match with those in the reference file (row where item="C"), and use -N to print records in the reference file that do not match with those in the input file (row where item="E").

```
$ more ref2.csv
item,cost
A,50
B,300
E,200
$ mnjoin k=item f=cost m=ref2.csv -n -N i=dat1.csv o=rsl2.csv
#END#
kgnjoin -N -n f=cost i=dat1.csv k=item m=ref2.csv o=rsl2.csv
$ more rsl2.csv
item%0,date,price,cost
A,20081201,100,50
A,20081213,98,50
B,20081002,400,300
B,20081209,450,300
C,20081201,100,
E,,,200
```

Related Commands

- `mjoin`: It is faster to use `mjoin` if the key in the reference file is unique.
- `mproduct`: Join combination of all records without using key. Each row in the reference file is joined with all records in the input data.
3.35 mnnormalize - Normalization

Specify the field at the \textit{f=} parameter, and specify the normalization method at \textit{c=} parameter.

**Format**

\texttt{mnnormalize c= f= [k=] [i=} [o=} [bufcount=] [-nfn] [-nfno] [-x] [-q] [precision=] [--help] [--version]}

**Parameters**

- \textit{c=} Specify the normalisation method listed as follows.
  - \texttt{z}: z score \( z_i = \frac{x_i - m}{u} \) \( (x_i: \text{number of data}, m: \text{arithmetic mean}, u: \text{standard deviation}) \)
  - \texttt{Z}: deviation value \( Z_i = 50 + 10 \times z_i \) \( \text{range}\): use linear conversion to transform minimum value 0 to maximum value 1 \( r_i = \frac{x_i - \text{min}_x}{\text{max}_x - \text{min}_x} \)
- \textit{f=} Specify the field to normalize here.
- \textit{k=} Key field name(s) \{aggregate key break processing\}
  - The key field specified is used as the unit for normalization.

**Examples**

**Example 1: Basic Example**

Normalize \( (z\text{ score}) \) \texttt{quantity} and \texttt{amount} field based on each \texttt{customer}, label the column names of the output as \texttt{qtyNominal} and \texttt{amtNominal} respectively.

```bash
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20

$ mnnormalize c=z k=customer f=quantity:qtyNominal,amount:amtNominal i=dat1.csv o=rsl1.csv

#END# kgnormalize c=z f=quantity:qtyNominal,amount:amtNominal i=dat1.csv k=customer o=rsl1.csv

$ more rsl1.csv
customer,quantity,amount,qtyNominal,amtNominal
A,1,10,-0.7071067812,-0.7071067812
A,2,20,0.7071067812,0.7071067812
B,1,15,-0.5773502692,0
B,3,10,1.154700538,-1
B,1,20,-0.5773502692,1
```

**Example 2: Deviation value**

```bash
$ mnnormalize c=Z k=customer f=quantity:qtyNominal,amount:amtNominal i=dat1.csv o=rsl2.csv

#END# kgnormalize c=Z f=quantity:qtyNominal,amount:amtNominal i=dat1.csv k=customer o=rsl2.csv

$ more rsl2.csv
customer,quantity,amount,qtyNominal,amtNominal
A,1,10,-0.7071067812,-0.7071067812
A,2,20,0.7071067812,0.7071067812
B,1,15,-0.5773502692,0
B,3,10,1.154700538,-1
B,1,20,-0.5773502692,1
```

```bash
$ mnnormalize c=Z k=customer f=quantity:qtyNominal,amount:amtNominal i=dat1.csv o=rsl2.csv

#END# kgnormalize c=Z f=quantity:qtyNominal,amount:amtNominal i=dat1.csv k=customer o=rsl2.csv

$ more rsl2.csv
customer,quantity,amount,qtyNominal,amtNominal
A,1,10,42.92893219,42.92893219
A,2,20,0.7071067812,0.7071067812
B,1,15,-0.5773502692,0
B,3,10,1.154700538,-1
B,1,20,-0.5773502692,1
```
Example 3: Linear transformation from 0 to 1

```
$ mnormalize c=range k=customer f=quantity:qtyNominal,amount:amtNorminal i=dat1.csv o=rsl3.csv
END
kgnormalize c=range f=quantity:qtyNominal,amount:amtNorminal i=dat1.csv k=customer o=rsl3.csv
$ more rsl3.csv
customer,quantity,amount,qtyNominal,amtNorminal
A,1,10,0,0
A,2,20,1,1
B,1,15,0,0.5
B,3,10,1,0
B,1,20,0,1
```

Related Command
3.36 mnrc_common - Select Records within Specified Range(s) from Reference File

Select the record in the input file that matches the records within the defined range(s) defined from the reference file. k= parameter specifies the key field name from the input file to match with the key defined in K= from the reference file. The selection criteria is based on the data series from the input file defined in r= parameter for records that falls within the data range in the reference file defined in the R= parameter. Add %n after the item name if the field defined at r= parameter is a numerical value.

Format


Parameters

k= Key item(s) to match in the input file (Multiple keys can be specified).
The key(s) specified will be matched with the key field(s) at K= parameter from the reference file.

m= Specify name of reference file.
Data is read from standard input if this parameter is not defined. (when i= is specified)

R= Define the value range (start,end) in the reference file.
If the first argument is NULL means less than, if the second argument is NULL means more than.

r= Field name of input file for range comparison. [%n]
Records in the input file that matches the key field specified in the k= parameter in the reference data is selected.
when processing as numeric value, %n will be added to field name defined at r= parameter.

K= Key field(s) in the reference data for matching (Multiple keys can be specified)
The key specified will be matched with the key field defined in k= parameter from the input file.
Records in the input file that matches the key field specified in the k= parameter in the reference data is selected.

u= Write unmatched records to this output file.

-r Reverse selection
Select records that is not within the data range defined at R= parameter.

Sort Criteria

Fields specified at r=,R= must be sorted beforehand. However, the numerical values defined in r=,R= should be sorted in ascending order to join with the numerical range. Where k=,K= is specified, the strings defined at the parameter must be sorted in ascending order.

For example, when the parameter k=key K=Key r=val%n R=range i=dat.csv m=ref.csv is specified, dat.csv data, should be sorted with msortf f=key,val%n as the criteria, and ref.csv data, should be sorted with msortf f=Key,range%n as the criteria.

Examples

Example 1: Basic Example

Select records where the transaction date is 20080203 with transaction "Amount" greater than 5 and less than 15 or greater than 40 and less than 50.

$ more dat1.csv
Date,Amount
20080123,10
20080203,10
20080203,20
20080203,45
20080410,50
$ more ref1.csv
Date,AmountF,AmountT
Example 2: Reverse selection

Add `-r` option to reverse selection criteria.

```
$ mnrcommon k=Date m=ref1.csv R=AmountF,AmountT r=Amount%n -r i=dat1.csv o=rsl2.csv
#END# kgnrcommon -r R=AmountF,AmountT i=dat1.csv k=Date m=ref1.csv o=rsl2.csv r=Amount%n
$ more rsl2.csv
Date,Amount
20080123,10
20080203,20
20080410,50
```

Related commands

- **mcommon**: Select common records in reference file
- **mnrjoin**: Natural join data from the reference file with multiple ranges.
3.37 mnrjoin - Natural Join within Multiple Ranges with Reference File

Join columns according to the range of values in the column from reference file. The field specified at r= parameter is matched with the the range of values defined as two arguments at the R= parameter in the reference file defined at the m= parameter. The field(s) specified at f= parameter are joined for records with the same value.

If there are more than one match for each record, natural join returns output for all rows. The range of values is compared as character strings by default. Attach %n after the field name at the r= parameter to process as numerical values.

Format


Parameters

- f= The field name(s) (multiple fields can be specified) to join from the reference file. When this is not defined, all fields except the key specified at K= will be joined.
- m= Reference file name.
- R= Field names of the range (limit to 2 fields).
  - Field names (start,end) of the range in reference file.
  - If the first field is NULL, the range is any number less than the ending value of the range.
  - If the second field is NULL, the range is any number greater than the starting value of the range.
- r= Compare the values in this field [%n] against the range.
  - Field name in the input file.
  - Add %n after the field name in the r= parameter to process as numerical values.
- k= Key field name(s) (multiple fields can be specified) from the input data for comparison.
  - Join records with same key fields in the input data k= and reference data K=.
- K= Key field name(s) (multiple fields can be specified) from the reference data for comparison.
  - This key field(s) from reference data is compared with the key field(s) from the input data specified at k= parameter, fields where records with common key are joined.
  - This parameter is not required if the field name name is the same as the one defined at the k= parameter.
- -n Output NULL values when reference data does not consist of input data.
- -N Output NULL values when input data does not consist of reference data.

For example, given the parameters k=key K=Key r=val%n R=range i=dat.csv m=ref.csv, if the sort criteria for the input data dat.csv is carried out by msortf f=key,val%n, the sort criteria for ref.csv should follow accordingly as msortf f=key,range%n.

Examples

Example 1: Basic Example

For records where the value of date field is 20080203, select those records in the input data where amount field is more than 5 but less than 15 and join field where avg=150. For records where amount field is more than 40 but less than 50, join field avg=200.

```
$ more dat1.csv
date,price
20080123,10
20080123,20
20080203,10
20080203,15
20080410,15
$ more ref1.csv
date,priceF,priceT,avg
```
Example 2: Output unmatched data

Use \(-n\) to return all records in the input data even if they do not match with those in the reference file (row where \texttt{avg= Null}), and use \(-N\) to return records in the reference file even if they do not match with those in the input file (rows where \texttt{price= null}). This is known as outer-join.

```
$ mnrjoin k=date f=avg m=ref1.csv R=priceF,priceT r=price%n i=dat1.csv o=rsl1.csv
#END# kgnrjoin R=priceF,priceT f=avg i=dat1.csv k=date m=ref1.csv o=rsl1.csv r=price%n
$ more rsl1.csv
date%0,price,avg
20080203,10,150
```

```
$ mnrjoin k=date f=avg m=ref1.csv R=priceF,priceT r=price%n -n -N i=dat1.csv o=rsl2.csv
#END# kgnrjoin -N -n R=priceF,priceT f=avg i=dat1.csv k=date m=ref1.csv o=rsl2.csv r=price%n
$ more rsl2.csv
date%0,price,avg
20080123,10,
20080123,20,
20080203,10,150
20080203,35,
20080203,,200
20080410,50,
```

Related Command

\texttt{mrjoin}: Use \texttt{mrjoin} if there are repeated values in join key (\texttt{K= field}) from the reference data.
3.38 mnullto - Replace NULL Values

Replace NULL values in the field(s) specified at f= parameter with a character string defined at v= parameter.

Format

mnullto f= [v=|-p] [O=} [-A] [i=] [o=} [-ufn] [-nfno] [-x] [--help] [--version]

Parameters

f= Replace null values in the field(s) (multiple fields can be specified).
v= Replace null values with this string.
-p Replace null values in the previous row.
   This option cannot be specified with v= parameter.
O= String to replace non-null values.
   When this parameter is not specified, non-null values will not be replaced.
-A Add replacement string as new column.
   When -A option is specified, define the new field name using a colon (:) after the field name.
Example: f=quantity:ReplacementFieldName.

Examples

Example 1: Basic Example

Replace NULL values in the verb—birthday—field with the string no value.

$ more dat1.csv
customer,birthday
A,19690103
B,
C,19500501
D,
E,
$ mnullto f=birthday v="no value" i=dat1.csv o=rsl1.csv
#END# kgnullto f=birthday i=dat1.csv o=rsl1.csv v=no value
$ more rsl1.csv
customer,birthday
A,19690103
B,no value
C,19500501
D,no value
E,no value

Example 2: Replace non-NULL values

Replace NULL values in the birthday field with the string "no value" and change non-null values to the string "value"—, and rename the output column as entry.

$ mnullto f=birthday:entry v="no value" O="value" i=dat1.csv o=rsl2.csv
#END# kgnullto O=value f=birthday:entry i=dat1.csv o=rsl2.csv v=no value
$ more rsl2.csv
customer,entry
A,value
B,no value
C,value
D,no value
E,no value
Example 3: Add new column

Replace Null values in the birthday field with the string "no value" and change non-null values to the string "value". Output the replacement strings in a new column named entry.

```bash
$ mnullto f=birthday:entry v="no value" O="value" -A i=dat1.csv o=rsl3.csv
#END# kgnullto -A O=value f=birthday:entry i=dat1.csv o=rsl3.csv v=no value
$ more rsl3.csv
customer,birthday,entry
A,19690103,value
B,,no value
C,19500501,value
D,,no value
E,,no value
```

Example 4: Replace values in previous row

```bash
$ more dat2.csv
id,date
A,19690103
B,
C,19500501
D,
E,
$ mnullto f=date -p i=dat2.csv o=rsl4.csv
#END# kgnullto -p f=date i=dat2.csv o=rsl4.csv
$ more rsl4.csv
id,date
A,19690103
B,19690103
C,19500501
D,19500501
E,19500501
```

Related Commands

`mdelnull`: Remove rows containing NULL values.

`mchgstr`: Replace NULL value with character strings.
3.39 mnumber - Serials

Show the alphabetical sequence (A,B,...,Z,AA,AB,...,AZ,BA,BB,...,ZZ,AAA,AAB,...) and save the output in a new column defined at \texttt{a=} parameter.

**Format**

```
mnumber a= [e=} [I=} [k=} [s=} [S=} [-B] [i=] [o=} [-nfn] [-nfno] [-x] [-q] [--help] [--version]
```

**Parameters**

- \texttt{a=} Specify the field name and the list of new serials.  
  [However, this parameter is not required when \texttt{-nfn} or \texttt{-nfno} options are specified]
- \texttt{e=} Process records with same Rank  
  Specify how to handle fields with same key same sort values.  
  Default setting is used (\texttt{e=seq}) with \texttt{No} \texttt{N} attached as the field name of the sequence, if the parameter is not specified.  
  \texttt{seq}: Create sequential serial numbers or alphabets for records with same rank.  
  \texttt{same}: Records with equal numerical or alphabetical values receive the same rank order,  
  \texttt{skip}: Records with equal numerical or alphabetical values receive the same rank order,  
  number of subsequent rank is skipped for the following record.  
  Note: \texttt{e={same/skip}} must be specified with the \texttt{s=} parameter.  
- \texttt{I=} Interval between the sequence.  
  However, a negative interval value cannot be specified for alphabet sequence.  
- \texttt{k=} Generate sequential characters for the key field(s) (multiple fields can be specified)  
- \texttt{s=} Specified field(s) (multiple fields can be specified) containing same rank values .  
  Note: This parameter must be declared with \texttt{e={same/skip}}.  
- \texttt{S=} Starting No  
  Specify the starting value of the sequence.  
  Uppercase alphabet letters for the alphabet sequence.  
- \texttt{-B} Assign same sequential number or alphabet to each key.  
  Records with the same key will each be assigned the same number (No) or alphabet.

**Examples**

**Example 1: Sequential numbers**

Generate sequential numbers for each value in ascending order in the \texttt{Customer} column. Name the sequence as \texttt{No} in a new column.

```
$ more dat1.csv  
Customer,Val,Sum  
A,29,300  
B,35,250  
C,15,200  
D,23,150  
E,10,100  
$ mnumber s=Customer a=No i=dat1.csv o=rsl1.csv  
#END# kgnumber a=No i=dat1.csv o=rsl1.csv s=Customer  
$ more rsl1.csv  
Customer%0,Val,Sum,No  
A,29,300,0  
B,35,250,1  
C,15,200,2  
D,23,150,3  
E,10,100,4
```
Example 2: Serialize the Date column

Sequentially number items in the Date column according to earliest date to latest date. Use same sequence number (No) for same Date. Save the sequence in a new column named "No".

```
$ more dat2.csv
Date
20090101
20090101
20090102
20090103
20090103

$ mnumber k=Date a=No -B i=dat2.csv o=rsl2.csv
#END# knumber -B a=No i=dat2.csv k=Date o=rsl2.csv
$ more rsl2.csv
Date,No
20090101,0
20090101,0
20090102,1
20090103,2
20090103,2
```

Example 3: Serialize the Sum column (use same alphabet for same Rank order)

Create a alphabetical sequence according to the Sum column which is arranged in descending order. Save the sequence in a new column named Rank. Assign the same alphabet character to items with the same values.

```
$ more dat3.csv
Customer,Val,Sum
A,3,300
B,1,250
C,2,250
D,1,150
E,1,100

$ mnumber a=Rank e=same s=Sum%nr S=A i=dat3.csv o=rsl3.csv
#END# knumber S=A a=Rank e=same i=dat3.csv o=rsl3.csv s=Sum%nr
$ more rsl3.csv
Customer,Val,Sum%nr,Rank
A,3,300,A
B,1,250,B
C,2,250,B
D,1,150,C
E,1,100,D
```

Example 4: Serialize the Sum column (sequential numbers for same Rank order)

Number records sequentially according to Sum column (sum arranged in descending order), and save serials in the "Rank" column. For items with same rank order, assign sequential numbers according to sort order.

```
$ mnumber a=Rank e=seq s=Sum%nr i=dat3.csv o=rsl4.csv
#END# knumber a=Rank e=seq i=dat3.csv o=rsl4.csv s=Sum%nr
$ more rsl4.csv
Customer,Val,Sum%nr,Rank
A,3,300,0
B,1,250,1
C,2,250,2
D,1,150,3
E,1,100,4
```

Example 5: Serialize the Sum column (Same No for same Rank)

Number records sequentially according to Sum column (sum arranged in descending order), and save the numbers in the Rank column. Assign the same No to records with the same Rank order.

```
$ mnumber a=Rank e=same s=Sum%nr i=dat3.csv o=rsl5.csv
#END# knumber a=Rank e=same i=dat3.csv o=rsl5.csv s=Sum%nr
$ more rsl5.csv
```
Example 6: Serialize the Sum column (duplicate numbers for same Rank and skip number for next record)

Number records sequentially according to Sum column (sum arranged in descending order), and save the numbers is the Rank column. Assign same RankNo number to records with same rank order, subsequent No is skipped for the following record.

```bash
$ mnumber a=Rank e=skip s=Sum%nr i=dat3.csv o=rsl6.csv
$ more rsl6.csv
```

```plaintext
Customer,Val,Sum%nr,Rank
A,3,300,0
B,1,250,1
C,2,250,1
D,1,150,2
E,1,100,3
```

Example 7: Number sequence starting from 10

Serialize the Sum column sequentially from 10 with items, where values of sum is arranged in ascending order. Save the serials in the "Score" column. Assign same RankNo to records with same Rank order, subsequent No is skipped for the following record.

```bash
$ more dat4.csv
$ mnumber a=Score e=same s=Sum%n S=10 i=dat4.csv o=rsl7.csv
$ more rsl7.csv
```

```plaintext
Customer,Val,Sum%nr,Score
A,1,100,10
B,1,150,15
C,1,250,20
D,2,250,20
E,3,300,25
```

Example 8: Start sequence from 10 with an interval of 5

Number the Sum column sequentially from 10 at an interval of 5, where values of sum is arranged in ascending order. Save the serials in the Score column. Assign the same number to records with the same Rank order.

```bash
$ mnumber a=Score e=same s=Sum%n S=10 I=5 i=dat4.csv o=rsl8.csv
$ more rsl8.csv
```

```plaintext
Customer,Val,Sum%nr,Score
A,1,100,10
B,1,150,15
C,1,250,20
D,2,250,20
E,3,300,25
```
Related Commands

- `mnewnumber`: Generate list of sequential numbers in a new dataset.
- `mbest`: Use `mnumber` if the query requires selection of records according to line numbers.
3.40 mpadding - Row Padding

Fill in values in between the records specified at f= parameter based on the key field specified at k= parameter. When v= parameter is specified, create padding records in between records with the specified string other than the fields specified at k=f=. Create padding with null values when -n option is specified. (Note: previous item value will be used as padding if both v= and -n parameters are not specified)

Format

mpadding [k=] f= [v=] [S=] [E=] [-n] [i=] [o=] [-nf] [-fn] [-xf] [-q] [--help] [--version]

Parameter

k= Specify the key field.

f= Target field name for continuous padding.
Specify the field name to create padding as continuous values between records.
When creating padding in numerical values, attach %n as no%n.
Attach %d when creating padding for dates, or attach %t for padding for time.
Attach %r when creating padding value in descending order as no%d%r.

v= Specify padding value as character string.
Create padding with with the specified string other than the fields specified at k=, f=.

S= Starting value
Specify the starting value of the series in f=.

E= Ending value
Specify the ending value of the series in f=.

-n Use null value as padding.
Return null values in fields other than those specified in k=, f=.

Examples

Example 1: Basic Example

Create padding with integer values (type=int) between records in no column. Insert 4, 5 between 3 and 6, and 7 between 6 and 8.

```
$ more dat1.csv
no
3
6
8
$ mpadding f=no%n i=dat1.csv o=rsl1.csv
#END# kgpadding f=no%n i=dat1.csv o=rsl1.csv
$ more rsl1.csv
no%0n
3
4
5
6
7
8
```

Example 2: Specify the starting and ending value

Insert padding between records as well as before and after the first and last records from the input data. Specify the starting and ending range at S=, E=.

```
$ mpadding f=no%n S=1 E=10 i=dat1.csv o=rsl2.csv
#END# kgpadding E=10 S=1 f=no%n i=dat1.csv o=rsl2.csv
$ more rsl2.csv
```
CHAPTER 3. COMMAND REFERENCE

Example 3: Padding with date

Create padding to fill in values between dates (type=date) in the date column. Create padding values in columns other than those specified at k=, f=.

```
$ more dat2.csv
date,dummy
20130929,a
20131002,b
20131004,c
$ mpadding f=date%d i=dat2.csv o=rsl3.csv
#END# kgpadding f=date%d i=dat2.csv o=rsl3.csv
$ more rsl3.csv
date%0,dummy
20130929,a
20130930,a
20131001,a
20131002,b
20131003,b
20131004,c
```

Example 4: Specify character string for padding

Specify the character string padding value at v=.

```
$ mpadding f=date%d v=padding i=dat2.csv o=rsl4.csv
#END# kgpadding f=date%d i=dat2.csv o=rsl4.csv v=padding
$ more rsl4.csv
date%0,dummy
20130929,a
20130930,padding
20131001,padding
20131002,b
20131003,padding
20131004,c
```

Example 5: Specify NULL value as padding character

NULL value can be used as padding when the -n option is specified.

```
$ mpadding f=date%d -n i=dat2.csv o=rsl5.csv
#END# kgpadding -n f=date%d i=dat2.csv o=rsl5.csv
$ more rsl5.csv
date%0,dummy
20130929,a
20130930,
20131001,
20131002,b
20131003,
20131004,c
```

Related Command
3.41 mpaste - Match and Merge Fields from Reference File

Merge input file with the reference file for matching rows. If data is different in size, merge with data with smaller size. It is also possible to match all data with different sizes with the inclusion of null values by specifying \(-n\) and \(-N\).

Format


Parameters

\(f=\) field name(s) (multiple fields can be specified) to merge with reference file.
All fields except the key field are merged.

\(m=\) Reference file name.
Read from standard input when this parameter is noted defined (when \(i=\) is specified).

\(-n\) Output NULL values when input data does not consist of reference data.

\(-N\) Output NULL values when reference data does not consist of input data.

Examples

Example 1: Basic Example

\begin{verbatim}
$ more dat1.csv
id1
id2
1
2
3
4
$ more ref1.csv
id2
a
b
c
d
$ mpaste m=ref1.csv i=dat1.csv o=rsl1.csv
#END# kgpaste i=dat1.csv m=ref1.csv o=rsl1.csv
$ more rsl1.csv
id1,id2
1,a
2,b
3,c
4,d
\end{verbatim}

Example 2: Example of merging data of different sizes

If the number of rows in the input file is different from the reference file, merge records according to the smaller file.

\begin{verbatim}
$ more ref2.csv
id2
a
b
$ mpaste m=ref2.csv i=dat1.csv o=rsl2.csv
#END# kgpaste i=dat1.csv m=ref2.csv o=rsl2.csv
$ more rsl2.csv
id1,id2
1,a
2,b
\end{verbatim}
Example 3: Outer join

If there are less number of rows in the reference file, NULL values will be assigned to records that did not match with the input file when `-n` option is specified.

```
$ mpaste m=ref2.csv -n i=dat1.csv o=rsl3.csv
#END# kgpaste -n i=dat1.csv m=ref2.csv o=rsl3.csv
$ more rsl3.csv
id1, id2
1, a
2, b
3,
4,
```

Example 4: Define fields to join

```
$ more ref3.csv
id2, val
a, R0
b, R1
c, R2
d, R3
$ mpaste f=val m=ref3.csv i=dat1.csv o=rsl4.csv
#END# kgpaste f=val i=dat1.csv m=ref3.csv o=rsl4.csv
$ more rsl4.csv
id1, val
1, R0
2, R1
3, R2
4, R3
```

Related Command

`mjoin`: Join using key field(s) if row numbers are not present.
3.42 mproduct - Cartesian Join with Reference File

Combine every row of column specified at f= parameter from the reference file at the m= parameter with every record from the input file.

Format


Parameters

f= Combine field name(s) (multiple fields can be specified) from reference file.
All field(s) are combine if this parameter is not specified.
m= Specify reference file name.
Read from standard input if this parameter is not defined (when i= is specified).

Examples

Example 1: Basic Example

Combine the \verb|date| column from reference file to the \verb|customer| column from the input file.

```
$ more dat1.csv
customer
A
B
$ more ref1.csv
date
20090101
20090201
20090301
$ mproduct f=date m=ref1.csv i=dat1.csv o=rsl1.csv
#END# kproduct f=date i=dat1.csv m=ref1.csv o=rsl1.csv
$ more rsl1.csv
customer,date
A,20090101
A,20090201
A,20090301
B,20090101
B,20090201
B,20090301
```

Related Command

mnjoin : Similar operation with mproduct but join key is specified.
3.43 mrand - Generate Random Numbers

Generate random number from the range 0.0 to 0.1, or generate random integers from a defined range. Define output column name at a= parameter.

This command uses Mersenne twister (developed in 1937) as pseudo random number generator. (Webpage of author, boost library).

Format

\texttt{mrand \{k=\} a= [max=] [min=] [S=} [\texttt{-int}] [i=} [o=} [-nfn] [-nfno] [-q] [---help] [---version]\

Parameters

\texttt{k=} \quad \text{Same random number is generated for same key value at the specified key field.}
\texttt{a=} \quad \text{New column name. [However, this parameter is not required when -nfn,-nfno option is specified]}
\texttt{max=} \quad \text{Maximum value of random number [default value: INT_MAX]}
\quad \text{Integer up to } 0\text{\texttt{\textasciitilde}FF\texttt{\textasciitilde}}2^{32} \{21 \text{ billion}\}.
\quad \text{\texttt{-int} must be specified with this parameter.}
\texttt{min=} \quad \text{Minimum value of random number [default=0].}
\quad \text{Integer up to } 0\text{\texttt{\textasciitilde}FF\texttt{\textasciitilde}}2^{32} \{21 \text{ billion}\}.
\quad \text{\texttt{-int} must be specified with this parameter.}
\texttt{S=} \quad \text{Random seed [default value: current time]}
\quad \text{The same random seed generates the same random number}
\quad \text{When S=} \text{is not specified, the default setting of random seed is set to the current time.}
\quad \text{Random seed value can be specified between -2147483648 \text{\texttt{\textasciitilde}2147483647}.}
\texttt{-int} \quad \text{Generate random integers.}

Examples

**Example 1: Basic example**

Generate random real numbers between 0.0 to 1.0.

```
$ more dat1.csv
Customer
A
B
C
D
E
$ mrand a=rand i=dat1.csv o=rsl1.csv
#END# kgrand a=rand i=dat1.csv o=rsl1.csv
$ more rsl1.csv
Customer,rand
A,0.6892393918
B,0.1042782064
C,0.07767942664
D,0.6829032891
E,0.9111980933
```

**Example 2: Basic Example 2**

Generate random integers with \texttt{-int}.

```
$ mrand a=rand -int i=dat1.csv o=rs12.csv
#END# kgrand -int a=rand i=dat1.csv o=rs12.csv
$ more rs12.csv
Customer,rand
A,646494551
B,1824612880
```
Example 3: Specify the minimum and maximum value of the random number

Generate a random number with a minimum value of 10 and maximum value of 100. Add the random numbers to a new column named rand.

```bash
$ mrand a=rand -int min=10 max=100 S=1 i=dat1.csv o=rsl3.csv
#END# kgrand -int S=1 a=rand i=dat1.csv max=100 min=10 o=rsl3.csv

$ more rsl3.csv
Customer,rand
A,47
B,100
C,75
D,94
E,10
```

Example 4: Generate random number by key

Given 4 customers A,B,C,D, same random number is generated for same customer.

```bash
$ more dat2.csv
Customer
A
A
A
B
B
C
D
D
D

$ mrand k=Customer -int min=0 max=1 a=rand i=dat2.csv o=rsl4.csv
#END# kgrand -int a=rand i=dat2.csv k=Customer max=1 min=0 o=rsl4.csv

$ more rsl4.csv
Customer%0,rand
A,0
A,0
A,0
B,0
B,0
C,0
D,0
D,0
D,0
```

Related Command

- **mselrand**: Select a random record.
- **mnewrand**: Generate new random dataset without using input file.
JOIN FROM REFERENCE FILE ACCORDING TO SPECIFIED RANGE

The value specified at r= parameter from the input data is matched with the range (value that falls above the first row and less than the next row) from the reference file, and subsequently joined with the value from the field specified at f= parameter. Use mnrjoin to join with complex conditions using range values. Consider using chgnum if there are not a lot of range.

Format

mrjoin r= [k=] [K=] [R=] [f=] [-n] [-lo] m=|i=|o=-|nf|[-nfn]|-x|[-q|--help] [--version]

Parameters

f= The field name(s) (multiple fields can be specified) to join from the reference file.
   When this is not defined, the all fields except the key specified at K= will be joined.

m= Reference file name.
   Read from standard input if this parameter is not set (when i= is specified).

r= Field name of the range for comparison [%n]
   Specify the field name from the input file.
   After the specified field is sorted (multiple fields can be specified), fields are joined.
   The value is interpreted as numeric range when %n is specified, otherwise,
   it is treated as character range.
   The specified field should not contain NULL values or the data may not be processed properly.

R= Field name containing range values in the reference file.
   When this parameter is not defined, the range is processed at r= parameter by default.

k= Key field name(s) (multiple fields can be specified) from the input data for comparison
   Join records with same key fields in the input data k= and reference data K=.

K= Key field name(s) (multiple fields can be specified) from the reference data for comparison
   This key field(s) from reference data is compared with the key field(s) from the input data specified at k= parameter,
   fields where records with common key are joined.
   This parameter is not required if the field name name is the same as the one defined at the k= parameter.

-n Output NULL values when reference data does not consist of input data.

-lo left-open interval
   The range with left open interval specified at R= parameter (greater than - below).

Examples

Example 1: Basic Example

Join category field low, middle,high to corresponding price range.

```
$ more dat1.csv
price
8
15
35
50
90
200
$ more ref1.csv
range,category
10,low
35,middle
80,high
100,

$ mrjoin r=price%n m=ref1.csv R=range f=category i=dat1.csv o=rsl1.csv
#END# kgrjoin R=range f=category i=dat1.csv m=ref1.csv o=rsl1.csv r=price%n
$ more rsl1.csv
price%0n,category
```
Example 2: Basic Example 2

```
$ mrjoin -lo r=price%n m=ref1.csv R=range f=category i=dat1.csv o=rsl2.csv
#END# kgrjoin -lo R=range f=category i=dat1.csv m=ref1.csv o=rsl2.csv r=price%n
$ more rsl2.csv
price%n,category
15,low
35,low
50,middle
90,high
```

Example 3: Basic Example 3

```
$ mrjoin -n r=price%n m=ref1.csv R=range f=category i=dat1.csv o=rsl3.csv
#END# kgrjoin -n R=range f=category i=dat1.csv m=ref1.csv o=rsl3.csv r=price%n
$ more rsl3.csv
price%n,category
15,low
35,middle
50,middle
90,high
```

Example 4: Join with different ranges for corresponding products

```
$ more dat2.csv
item,price
A,10
A,20
B,10
B,20
$ more ref2.csv
item,price,category
A,10,low
A,15,high
A,100,
B,10,low
B,35,high
B,100,
$ mrjoin k=item r=price%n m=ref2.csv f=category i=dat2.csv o=rsl4.csv
#END# kgrjoin f=category i=dat2.csv k=item m=ref2.csv o=rsl4.csv r=price%n
$ more rsl4.csv
item%0,price%1n,category
A,10,low
A,20,high
B,10,low
B,20,low
```

Related Commands

- **mchgnum**: Specify a number range to replace / add value.
- **mjoin**: Use this command to join matching strings instead of using numeric range.
- **mnrcommon**: Use this command to select records rather than to join fields.
CHAPTER 3. COMMAND REFERENCE

3.45 msed - Replace String Matching Regular Expression

Replace string in the fields specified in the \texttt{f=} parameter with a string specified in the \texttt{v=} parameter for content that matches the regular expression specified in the \texttt{c=} parameter.

Format

\texttt{msed} \texttt{c=} \texttt{f=} \texttt{v=} \texttt{[-A]} \texttt{[-g]} \texttt{[-W]} \texttt{[i=} \texttt{[o=} \texttt{[-nfn]} \texttt{[-nfno]} \texttt{[-x]} \texttt{[--help]} \texttt{[--version]}

Parameters

\texttt{f=} specify the target list of field name(s) (multiple fields can be specified) for parsing.

\texttt{c=} Define the regular expression for string substitution.
Refer to usage of regular expressions.

\texttt{v=} Specify the string to replace the substring that matches with the regular expression specified in the \texttt{c=} parameter.

It is possible to substitute match result with the following methods:

\texttt{$\&$} : Matched string
\texttt{$\&$} : Search for the string from the beginning of the target replacement character string, until a string is matched.
\texttt{$\&$} : After a matched string, substitute target replacement string with matched string till the end.
\texttt{\$N} : partial string match for the \texttt{N}-th occurrence (\texttt{N}>=1).
\texttt{-A} Instead of replacing the specified field, add field as a new column.
\texttt{-g} Replace all matches of the regular expression.
\texttt{-W} Replace wide character matches of the regular expression.

Using regular expressions

List of regular expression specified in the \texttt{c=} parameter is shown from Table 3.10 to Table 3.13.

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Description</th>
<th>Example of pattern</th>
<th>Example of \texttt{c=,v=}</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Any character</td>
<td>abbbcc</td>
<td>\texttt{c=, v=X} \texttt{-g}</td>
<td>XXXXX</td>
</tr>
<tr>
<td>[abc]</td>
<td>either a, b, or c character</td>
<td>abbbcc</td>
<td>\texttt{c=[ac]} \texttt{v=X} \texttt{-g}</td>
<td>XbbXX</td>
</tr>
<tr>
<td>[^abc]</td>
<td>Any character other than a, b, c</td>
<td>abbbcc</td>
<td>\texttt{c=[^ac]} \texttt{v=X} \texttt{-g}</td>
<td>xXXcc</td>
</tr>
<tr>
<td>[a-z]</td>
<td>Any character from a to z</td>
<td>abbbcc</td>
<td>\texttt{c=[a-b]} \texttt{v=X} \texttt{-g}</td>
<td>XXXcc</td>
</tr>
<tr>
<td>[^a-z]</td>
<td>Any character outside the range of a to z</td>
<td>abbbcc</td>
<td>\texttt{c=[^a-b]} \texttt{v=X} \texttt{-g}</td>
<td>abbbXX</td>
</tr>
<tr>
<td>\t</td>
<td>Tab character</td>
<td>ab#cd&amp;ef</td>
<td>\texttt{c=\t} \texttt{v=X} \texttt{-g}</td>
<td>XX#XX XX</td>
</tr>
<tr>
<td>\w</td>
<td>Word string ([0-9a-zA-Z_])</td>
<td>ab#cd&amp;ef</td>
<td>\texttt{c=\w} \texttt{v=X} \texttt{-g}</td>
<td>abXcdXef</td>
</tr>
<tr>
<td>\s</td>
<td>Characters other than Word string</td>
<td>ab#cd&amp;ef</td>
<td>\texttt{c=\s} \texttt{v=X} \texttt{-g}</td>
<td>abXcdXef</td>
</tr>
<tr>
<td>\S</td>
<td>Non-whitespace character</td>
<td>ab cd ef</td>
<td>\texttt{c=\S} \texttt{v=X} \texttt{-g}</td>
<td>abXcdXef</td>
</tr>
<tr>
<td>\d</td>
<td>Numeric constituent characters ([0-9])</td>
<td>ab12c0</td>
<td>\texttt{c=\d} \texttt{v=X} \texttt{-g}</td>
<td>abXXcX</td>
</tr>
<tr>
<td>\D</td>
<td>Non-numeric constituent characters</td>
<td>ab12c0</td>
<td>\texttt{c=\D} \texttt{v=X} \texttt{-g}</td>
<td>XX12X0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Description</th>
<th>Example of pattern</th>
<th>Example of \texttt{c=,v=}</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a*</td>
<td>Zero or more repetition of a</td>
<td>abbbcc</td>
<td>\texttt{c=a*} \texttt{v=X}</td>
<td>Xcc</td>
</tr>
<tr>
<td>a+</td>
<td>Repetition of one or more a</td>
<td>abbbcc</td>
<td>\texttt{c=a+} \texttt{v=X}</td>
<td>Xcc</td>
</tr>
<tr>
<td>a?</td>
<td>Single occurrence of a</td>
<td>abbbcc</td>
<td>\texttt{c=a?} \texttt{v=X}</td>
<td>Xbbcc</td>
</tr>
<tr>
<td>a{M,N}</td>
<td>Repetition of a more than M and less than N</td>
<td>abbbbbcc</td>
<td>\texttt{c=a{3,4}} \texttt{v=X}</td>
<td>Xbbcc</td>
</tr>
<tr>
<td>a{M}</td>
<td>Repetition of a more than M times</td>
<td>abbbbc</td>
<td>\texttt{c=a{3}} \texttt{v=X}</td>
<td>Xbbcc</td>
</tr>
<tr>
<td>a \mid b</td>
<td>a or b</td>
<td>abbc</td>
<td>\texttt{c=(a \mid (bc))} \texttt{v=X}</td>
<td>XbbX</td>
</tr>
<tr>
<td>?</td>
<td>Shortest match after the repeat sign</td>
<td>abbc</td>
<td>\texttt{c=a*?} \texttt{v=X}</td>
<td>Xbbbc</td>
</tr>
</tbody>
</table>
Table 3.12: Position of regular expression

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Description</th>
<th>Example of pattern</th>
<th>Example of c=,v=</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Match from the beginning</td>
<td>abac</td>
<td>c=^a v=X -g</td>
<td>Xbac</td>
</tr>
<tr>
<td>$</td>
<td>Match till the end</td>
<td>acac</td>
<td>c=c$ v=X -g</td>
<td>acaX</td>
</tr>
<tr>
<td>\b</td>
<td>Match starting characters of string</td>
<td>aac ba ac bac</td>
<td>c=\ba v=X -g</td>
<td>Xac bX Xc bac</td>
</tr>
<tr>
<td>\B</td>
<td>Match within the string</td>
<td>aac ba ac bac</td>
<td>c=\Ba v=X -g</td>
<td>aXc ba ac bXc</td>
</tr>
</tbody>
</table>

Table 3.13: Others

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Description</th>
<th>Example of pattern</th>
<th>Example of c=,v=</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(expr)</td>
<td>Grouping</td>
<td>abbcabbc</td>
<td>c=(ab)(bc)\1 v=x</td>
<td>Xabc</td>
</tr>
<tr>
<td>\1,..,\9</td>
<td>Back reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(?=expr)</td>
<td>Position before matched string at expr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(?\1 expr)</td>
<td>Position before unmatched string at expr</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples

Example 1: Basic Example

Replace the 4-digit substring in the zipCode field starting 00 with ####.

```
$ more dat1.csv
customer,zipCode
A,6230041
B,6240053
C,6330032
D,6230087
E,6530095
```

```
$ msed f=zipCode c=00.. v=#### i=dat1.csv o=rsl1.csv
#END# kgsed c=00\d\d f=zipCode i=dat1.csv o=rsl1.csv v=####
$ more rsl1.csv
customer,zipCode
A,623####
B,624####
C,633####
D,623####
E,653####
```

Example 2: Specify field name

Replace the 4-digit substring in the zipCode field starting 00 with ####. Save output in column zipCode4.

```
$ msed f=zipCode:zipCode4 c='00\d\d' v=#### i=dat1.csv o=rsl2.csv
#END# kgsed c=00\d\d f=zipCode:zipCode4 i=dat1.csv o=rsl2.csv v=####
$ more rsl2.csv
customer,zipCode4
A,623####
B,624####
C,633####
D,623####
E,653####
```

Example 3: Global replacement

Global search using the regular expression - to replace value of 0 in zipCode.

```
$ msed f=zipCode c=0 v=- -g i=dat1.csv o=rsl3.csv
#END# kgsed -g c=0 f=zipCode i=dat1.csv o=rsl3.csv v=-
$ more rsl3.csv
customer,zipCode
A,623--41
B,624--53
C,633--32
```
Example 4: Replace substring

Delete fruit from the beginning of the string in item. Note that when first match (\(^\)) is specified, the substring within the word grapefruit in the last row is retained.

```
$ more dat2.csv
item,price
fruit:apple,100
fruit:peach,250
fruit:pineapple,300
fruit:orange,450
fruit:grapefruit,500
$ msed f=item c='^fruit' v= -g i=dat2.csv o=rsl4.csv
#END# kgsed -g c=^fruit f=item i=dat2.csv o=rsl4.csv v=
$ more rsl4.csv
item,price
:apple,100
:peach,250
:pineapple,300
:orange,450
:grapefruit,500
```

Example 5: Substitution using match results

Replaced 1 or more consecutive character strings of \(b\) using \(\$&\) is defined in the \(v=\).

```
$ more dat3.csv
str1
abc
abbc
ac
$ msed f=str1 c='b+' v='#$\&' i=dat3.csv o=rsl5.csv
#END# kgsed c=b+ f=str1 i=dat3.csv o=rsl5.csv v=#$\&#
$ more rsl5.csv
str1
a#b#c
a#bb#c
ac
```

Example 6: Combination of the global match

When performing a global match, each match is evaluated against the contents defined at \(v=\).

```
$ msed f=str1 c=b v='##&' -g i=dat3.csv o=rsl6.csv
#END# kgsed -g c=b f=str1 i=dat3.csv o=rsl6.csv v='##&'
$ more rsl6.csv
str1
a#b#c
a#bbb#c
ac
```

Example 7: Prefix substitution

Replace the matching first character of \(b\) in the character string (prefix) using \(\$'\).

```
$ msed f=str1 c=b v='##' i=dat3.csv o=rsl7.csv
#END# kgsed c=b f=str1 i=dat3.csv o=rsl7.csv v='##'
$ more rsl7.csv
str1
a#a#c
a#a#bc
ac
```
Example 8: Suffix substitution

Replace the matching last character of b in the character string (suffix) using $'$.  

```
$ msed f=str1 c=b v="#$" i=dat3.csv o=rsl8.csv
END# kgsed c=b f=str1 i=dat3.csv o=rsl8.csv v="#$'#
$ more rsl8.csv
str1
a#c#c
a#bc#bc
ac
```

Related Commands

- **mchgstr**: Use this command to replace with a simple string match.
- **mcal**: Include several functions to handle the regular expression.
3.46  **msel - Select Records with Conditions**

Define the computation criteria at \( c= \) parameter, the record is selected if condition returns true. All operators and functions available in mcal command can be used in the conditional function. For more details, please refer to mcal.

**Format**

```
msel  c=  [u=}  [-r]  [i=}  [o=}  [-nfn]  [-nfn]  [-x]  [--help]  [--version]
```

**Parameters**

- **c=** Define the expression using combinations of operators and functions. Refer to mcal for more details.
- **o=** Records matching the condition will be printed to this output file.
- **u=** Records that do not match the condition will be printed to this output file.
- **-r** Reverse selection. Select records excluded from the selection condition defined in \( c= \)

**Examples**

**Example 1: Basic example**

Select records where "Amount" is greater than 40. Write the unmatched records to a different output file file unmatch1.csv.

```
$ more dat1.csv
Customer,Quantity,Amount
A,1,10
A,2,20
B,1,30
B,3,40
B,1,50
$ msel c='${Amount}>40' u=unmatch1.csv i=dat1.csv o=match1.csv
#END# kgsel c=${Amount}>40 i=dat1.csv o=match1.csv u=unmatch1.csv
$ more match1.csv
Customer,Quantity,Amount
B,1,50
$ more unmatch1.csv
Customer,Quantity,Amount
A,1,10
A,2,20
B,1,30
B,3,40
```

**Example 2: Selecting records with null value(s)**

No records will be selected when the condition defined \( c= \) returned a null value. Records that do not match the condition will be written to a separate file defined in \( u= \).

In the following example, the first three rows of data from column b are -1, null, and 1. When selecting records where \( b \) is greater than 0, the query record with a null value will be treated as an exception saved in the unmatched records file.

```
$ more dat2.csv
a,b
A,-1
B,
C,1
$ msel c='${b}>0' i=dat2.csv o=match2.csv u=unmatch2.csv
#END# kgsel c=${b}>0 i=dat2.csv o=match2.csv u=unmatch2.csv
```
Example 3: Specify -r option

Null value is always evaluated as a unknown value regardless of the condition. Thus, records with null value is not selected.

In the following example, the reverse selection parameter -r is used with the same condition in the previous example. Even though the selection criteria is inverted, the query record with a null value will be treated as an exception saved in the unmatched records file as in the previous example.

```
$ msel -r c='${b}>0' i=dat2.csv o=match3.csv u=unmatch3.csv
$ more match3.csv
a,b
A,-1
$ more unmatch3.csv
a,b
B,
C,1
```

Related Commands

- **mselnum**: Select records with simple numeric range.
- **mselstr**: Select records matching query string
- **mcal**: Return the calculated results instead of selecting records.
CHAPTER 3. COMMAND REFERENCE

3.47 mselnum - Select Records Matching Range

This command selects records that matches the range specified at c= with the values in the column specified at the f= parameter. Various selection criteria can be specified as arguments in the parameters as follows. Use msel command to query complex selection criteria such as matching against a combination of string characters. For more information about OR, AND conditions for key field, refer to mselstr command.

- Range types include open range, closed range, inclusive and exclusive bounds and infinite range.
- c=specify multiple range and select records that matches either range (OR condition).
- f=specify multiple fields at which the values matches either range (OR condition).
- f=change the logical operator from OR to AND with the -F option.
- k=specify key value as the selection unit.
- Use AND logical operator to select records based on the key value with the -R option.

Typical examples are shown in 3.14 to 3.17.

<table>
<thead>
<tr>
<th>Table 3.14: Input Data</th>
<th>Table 3.15: Select rows if the values in the val column falls on the range of 1 to 3 mselnum f=val c='[1,3]'</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>val</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>-3</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>6</td>
</tr>
</tbody>
</table>

| Table 3.16: Select rows in val field with values greater than or Table 3.17: Select rows with values greater equal to 1 but less than or equal to 5 mselnum f=val c='[1,3)' |
|----------------------|---------------------------------------------------------------|
| key | val |                                                      |
| a   | 1   |                                                      |

| Table 3.18: Select rows less than or equal to 1 or greater than or equal to 5 mselnum f=val c='(1,][5,)' |
|----------------------|---------------------------------------------------------------|
| key | val |                                                      |
| a   | 1   |                                                      |
| a   | -3  |                                                      |
| b   | 6   |                                                      |

Format

mselnum f= c= [k=} [u=] [-F] [-x] [-R] [i=] [o=] [bufcount=] [-nf] [-nfo] [-x] [-q] [--help] [--version]
f= Field name of target query (allow multiple fields).
c= Select row(s) where the data array specified at f= parameter matches with the specified range at this parameter (allow multiple ranges).
k= Key field(s) as unit of selection (Multiple range can be specified).
o= Records matching the condition will be printed to this output file.
u= Records that do not match the condition will be printed to this output file.
-f Select all records that matches the defined value if multiple items are defined at f= parameter.
-r Reverse selection
  Select records excluded from the selection condition.
-R Select if all rows with the same key specified at k= parameter matches the criteria.

Parameters

Examples

Example 1: Basic Example

Select rows where val is greater than 2 and below 5, i.e. records matching id=2,5 are selected.

```bash
$ more dat1.csv
id,val
1,5.1
2,5
3,-2.0
4,5,2.0
$ mselnum f=val c='[2,5]' i=dat1.csv o=rsl1.csv
#END# kgselnum c=[2,5] f=val i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val
2,5
5,2.0
```

Example 2: Greater than range

Select rows where val is greater than 2, i.e. records where id=1,2,5.

```bash
$ mselnum f=val c='[2,]' i=dat1.csv o=rsl2.csv
#END# kgselnum c=[2,] f=val i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val
1,5.1
2,5
5,2.0
```

Related Commands

- **msel**: Selection of records by conditional expressions.
- **mselstr**: Selection by string matching.
### 3.48 mselrand - Random Sampling

Random selection of records based on the number of rows set at \( c = \) and \( p = \) parameters (random sampling without replacement). When \( k = \) is specified, a defined number of records with same key are randomly selected, when option \(-B\) specified at the same time, records are selected based in the key.

This command used Mersenne twister (developed in 1937) as pseudo random number generator (Webpage of author, boost library).

**Format**

```bash
mselrand c=|p= [k=} [S=} [u=} [-B] [i=} [o=} [-nfn] [-nfno] [-x] [-q] [--help] [--version]
```

**Parameters**

- **c=** Select row(s) based on the number of keys and field specified. This parameter must be specified when \( p = \) is not specified.
- **p=** Define the percentage of records for selection based on each key value. This parameter must be specified if \( c = \) parameter is not specified.
- **k=** Select certain number of rows randomly from records with same key (Allow multiple fields).
- **S=** The same random seed generates the same row selection sequence.
  - The default setting of random seed is set to the current time if the random seed is not specified.
  - Range of random seed value is between -2147483648 - 2147483647.
- **u=** Print unmatched records to this output file.
- **-B** Selection based on key unit.

**Examples**

**Example 1: Basic Example**

Randomly select 1 transaction for each customer.

```bash
$ more dat1.csv
Customer,Date,Amount
A,20081201,10
A,20081207,20
A,20081213,30
B,20081002,40
B,20081209,50
$ mselrand k=Customer c=1 S=1 i=dat1.csv o=rsl1.csv
#END# kgselrand S=1 c=1 i=dat1.csv k=Customer o=rsl1.csv
$ more rsl1.csv
Customer%0,Date,Amount
A,20081201,10
B,20081002,40
```

**Example 2: Randomly select a percentage of records**

Select 50% of each customers’ records at random. Save other records to a separate file oth.csv.

```bash
$ mselrand k=Customer p=50 S=1 u=oth2.csv i=dat1.csv o=rsl2.csv
#END# kgselrand S=1 i=dat1.csv k=Customer o=rsl2.csv p=50 u=oth2.csv
$ more rsl2.csv
Customer%0,Date,Amount
A,20081201,10
B,20081002,40
$ more oth2.csv
Customer%0,Date,Amount
A,20081201,10
B,20081209,50
```
Example 3: Select records by same key

In the following example, select two out of the four customers A, B, C, D at random. Customer C, D is selected, and all records of customer C, D is printed to the output.

```
$ more dat2.csv
Customer,Date,Amount
A,20081201,10
A,20081207,20
A,20081213,30
B,20081002,40
B,20081209,50
C,20081210,60
D,20081201,70
D,20081205,80
D,20081209,90
$ mselrand k=Customer c=2 S=1 -B i=dat2.csv o=rsl3.csv
#END# kgselrand -B S=1 c=2 i=dat2.csv k=Customer o=rsl3.csv
$ more rsl3.csv
Customer%0,Date,Amount
C,20081210,60
D,20081201,70
D,20081205,80
D,20081209,90
```

Related Commands

- msel: Use normally distributed random numbers.
- mrand: Add random numbers as a new column.
3.49 mselstr - Select Records Matching Query String

For records where the values in fields \( f = \) match with the string specified at \( v = \) are selected.

Commonly used examples are shown in 3.19 - 3.21. In table 3.20, select records where \( val \) is "y" regardless of the value of \( key \). In table 3.21, if any of the record contains the value "x" in \( val \), records with the same key value will be selected. i.e. all records with value "a" in \( key \) column are selected. Since none of the records with key value "b" contains value "x", none records are selected.

Table 3.19: Input data

<table>
<thead>
<tr>
<th>key</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>a</td>
<td>y</td>
</tr>
<tr>
<td>b</td>
<td>y</td>
</tr>
<tr>
<td>b</td>
<td>z</td>
</tr>
</tbody>
</table>

Table 3.20: \( f = val \ v = y \)

<table>
<thead>
<tr>
<th>key</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>y</td>
</tr>
<tr>
<td>b</td>
<td>y</td>
</tr>
</tbody>
</table>

Table 3.21: \( k = key \ f = val \ v = x \)

<table>
<thead>
<tr>
<th>key</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>x</td>
</tr>
<tr>
<td>a</td>
<td>y</td>
</tr>
</tbody>
</table>

Various selection criteria can be carried out with the parameters below. Use msel command to build complex conditions using regular expressions and operators which cannot be specified in this command.

- \( v = \) Match any character string from the list of string(s) specified.
- \( f = \) Match character string from the column(s) specified.
- AND operator can be used to match values multiple fields (-F option).
- Matching for exact, start, middle or partial string can be specified (-head, -tail, -sub option).
- \( k = \) Select records related to the defined key.
- Select records that matches all conditions by the key field (-R option).

Sample data with same key, containing two records and two columns is shown in (Table 3.22).

```
mselstr k=key f=fld1,fld2 v=s1,s2
```

Matching criteria without -R, -F options are shown in 3.23.

Table 3.22: Input data

<table>
<thead>
<tr>
<th>key</th>
<th>fld1</th>
<th>fld2</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>( v_{a1} )</td>
<td>( v_{a2} )</td>
</tr>
<tr>
<td>k</td>
<td>( v_{b1} )</td>
<td>( v_{b2} )</td>
</tr>
</tbody>
</table>

Table 3.23: Using the input data shown in Table 3.22, the query results of the command mselstr k=key f=fld1,fld2 v=v1,v2 with and without -R and F options differs accordingly. If query matches all conditions, the output will print all rows (2 rows), when there is no matched records, no records will returned.

<table>
<thead>
<tr>
<th>-F</th>
<th>-R</th>
<th>Matching conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-F</td>
<td>((v_{a1} = s1 \text{ or } v_{a1} = s2) \text{ or } (v_{a2} = s1 \text{ or } v_{a2} = s2)) or ((v_{b1} = s1 \text{ or } v_{b1} = s2) \text{ or } (v_{b2} = s1 \text{ or } v_{b2} = s2)))</td>
</tr>
<tr>
<td>-F</td>
<td>-R</td>
<td>((v_{a1} = s1 \text{ or } v_{a1} = s2) \text{ and } (v_{a2} = s1 \text{ or } v_{a2} = s2)) or ((v_{b1} = s1 \text{ or } v_{b1} = s2) \text{ or } (v_{b2} = s1 \text{ or } v_{b2} = s2))</td>
</tr>
<tr>
<td></td>
<td>-F</td>
<td>((v_{a1} = s1 \text{ or } v_{a1} = s2) \text{ and } (v_{a2} = s1 \text{ or } v_{a2} = s2))</td>
</tr>
</tbody>
</table>

Format

```
```
### Parameters

#### f=
Target field name for query (allow multiple fields).

#### v=
Select rows where the string specified at f= parameter matches any of the specified string(s) (allow multiple fields).

#### k=
Select records based on the defined key field (allow multiple fields).

#### o=
Print record(s) matching query to specified output file.

#### u=
Print unmatched record(s) to this output file.

#### -F
Match all character strings specified at the f= parameter.

#### -r
Reverse selection
Remove record matching records.

#### -R
Returns rows that match all character strings specified at the k= parameter.

#### -sub
Search for substring that matches the part of the string pattern.
Select records based on substring match specified in v= parameter, against the string specified column(s) at the f= parameter.

#### -head
Match beginning of string
- tail
Match end of string

#### -W
Match a sequence of wide-character substring when -sub,-head,-tail option is specified.

### Examples

#### Example 1: Basic example
Select records matching apple and orange in the Product field, print matching results to rsl1.csv file. Unmatched records such as pineapplejuice will be saved to other.csv file using the parameter u=oth1.csv.

```bash
$ more dat1.csv
Product,Amount
apple,100
milk,350
orange,100
pineapplejuice,500
wine,1000
$ mselstr f=Product v=apple,orange u=oth1.csv i=dat1.csv o=rsl1.csv
#END# kgselstr f=Product i=dat1.csv o=rsl1.csv u=oth1.csv v=apple,orange
$ more rsl1.csv
Product,Amount
apple,100
orange,100
$ more oth1.csv
Product,Amount
milk,350
pineapplejuice,500
wine,1000
```

#### Example 2: Remove records
Contrary to example 1, remove records matching keywords apple and orange using the -r option, the output is saved to rsl2.csv file.

```bash
$ mselstr f=Product v=apple,orange -r i=dat1.csv o=rsl2.csv
#END# kgselstr -r f=Product i=dat1.csv o=rsl2.csv v=apple,orange
$ more rsl2.csv
Product,Amount
apple,100
orange,100
$ more oth1.csv
Product,Amount
milk,350
pineapplejuice,500
wine,1000
```

#### Example 3: Select based on the key unit
Select all records of customer who have purchased oranges by specifying Customer at the k= parameter. Save unmatched records to oth2.csv.

```bash
$ more dat2.csv
Customer,Product,Amount
A,apple,100
```

```bash
```
Example 4: Partial match

Select records where the Product field contain the keyword apple, and save the output to a file named rsl4.csv. Records with partial match such as pine(apple)juice will also be saved in the output file rsl4.csv.

```
$ mselstr f=Product v=apple -sub i=dat1.csv o=rsl4.csv
#END# kgselstr -sub f=Product i=dat1.csv o=rsl4.csv v=apple
$ more rsl4.csv
Product,Amount
apple,100
pineapplejuice,500
```

Example 5: Wide character substring match

Select records where the Product field contains wide characters "柿", "桃", and "葡萄".

Matching may be based on single byte character if the query string includes wide character, the query string may be interpreted as multibyte character for matching. Therefore, it is necessary to indicate wide character in the query string with -W option.

```
$ more dat3.csv
Product,Amount
fruit:柿,100
fruit:桃,250
fruit:葡萄,300
fruit:梨,450
fruit:梅,500
$ mselstr f=Product v=柿,桃,葡萄 -sub -W i=dat3.csv o=rsl5.csv
#END# kgselstr -W -sub f=Product i=dat3.csv o=rsl5.csv v=柿,桃,葡萄
$ more rsl5.csv
Product,Amount
fruit:柿,100
fruit:桃,100
fruit:梅,250
fruit:葡萄,300
```

Example 6: Select product(s) with consecutive purchases in 2013.

Use the -F option to select transactions where the date of purchase and the previous date of purchase for the product both took place in 2013. Save the query results to an output file rsl6.csv. Save unmatched records to oth3.csv.

```
$ more dat4.csv
Customer,Product,Amount,Gender,Date,PreviousDate
A,apple,100,1,2013/01/04,2013/01/01
A,milk,350,1,2013/04/04,2011/05/06
```

Example 3: Data manipulation and query operations
Example 7: Extract all transactions of customers who have consecutive purchases in 2013

Use the \( k \) parameter to select all transactions of customers who have purchased a product with date of purchase and date of previous purchase both took place in 2013. Save unmatched records to a file `oth4.csv`.

$$ \texttt{mselstr k=Customer f=Date,PreviousDate} \quad \texttt{-F -sub v=2013 u=oth4.csv i=dat4.csv o=rsl7.csv} $$

```
Customer%0,Product,Amount,Gender,Date,PreviousDate
A,apple,100,1,2013/01/04,2013/01/01
A,milk,350,1,2013/04/04,2011/05/06
B,orange,100,2,2012/11/11,2011/11/11
B,pineapple,500,2,2013/04/15,2013/04/01
B,wine,1000,2,2012/12/24,2011/12/24
C,apple,100,2,2013/02/14,2013/02/01
C,orange,100,2,2013/02/14,2013/01/31
$ more oth4.csv
```

Example 8: Select new customer(s) who purchased in 2013

Use the \(-R\) option to select all transactions of new customer(s) who made their first purchase in 2013, where date of previous purchase is NULL. Write the query results to an output file `rsl8.csv`, and save unmatched records to `oth5.csv`.

$$ \texttt{mselstr k=Customer f=Date,PreviousDate} \quad \texttt{-F -R -sub v=2013 NULL u=oth5.csv i=dat4.csv o=rsl8.csv} $$

```
Customer%0,Product,Amount,Gender,Date,PreviousDate
C,apple,100,2,2013/02/14,2013/01/31
$ more oth5.csv
```

```
Customer%0,Product,Amount,Gender,Date,PreviousDate
A,apple,100,1,2013/01/04,2013/01/01
A,milk,350,1,2013/04/04,2011/05/06
B,orange,100,2,2012/11/11,2011/11/11
B,pineapple,500,2,2013/04/15,2013/04/01
B,wine,1000,2,2012/12/24,2011/12/24
D,orange,100,2,2011/10/28,2011/10/28
$ more rsl8.csv
```
Related Commands

msel : Select records with more complex criteria.

mcommon : When selecting a large number of target strings use mcommon command.
### 3.50 msep - Partition Records

Define the file name and location the save the separate records at the `d=` parameter. Since it is possible to embed the `${fieldname}` in the file name specified, as a result, the data can be split into separate files for corresponding fields. For example, the argument `d=./out/${date}.csv` specifies the `out` directory in the current directory, and the file is created from corresponding values in the `date` field.

#### Format

```
```

#### Parameters

- **d=** Specify the field name used for splitting to different data files. String specify here will be added as file name for each record. Embed field name as `${field name}`.
- **-p** Create the new directory name specify at the `d=` parameter which does not currently exist.

#### Examples

**Example 1: Basic Example**

Create a directory named `dat`, and separate the data according to the `date` value in the directory.

```
$ more dat1.csv
item,date,quantity,price
A,20081201,1,10
B,20081201,4,40
A,20081202,2,20
A,20081203,3,30
B,20081203,5,50
$ msep d=./dat/${date}.csv -p i=dat1.csv
#END# kgsep -p d=./dat/${date}.csv i=dat1.csv
$ ls ./dat
20081201.csv
20081202.csv
20081203.csv
$ more ./dat/20081201.csv
item,date,quantity,price
A,20081201,1,10
B,20081201,4,40
$ more ./dat/20081202.csv
item,date,quantity,price
A,20081202,2,20
$ more ./dat/20081203.csv
item,date,quantity,price
A,20081203,3,30
B,20081203,5,50
$ more ./dat/B.csv
./dat/B.csv: No such file or directory
```

#### Related Commands

- **msep2**: While the functionality is similar with `msep`, serial number is used to name the output file, and prints a list of corresponding key and file name to a separate file.
- **mcat**: This command restore and merge all the partitioned files by `sep` to the original file.
3.51 msep2 - Separate Records And Return Fields Table

Separate the data according to the value of the field(s) specified in k=. Partitioned data is automatically stored in numbered file name sequence. A table is created with the list of keys specified at k= and the corresponding file name for each key.

Format

msep2 k= O= a= [-p] [i=} [o=} [-nfn] [-nfno] [-x] [-q] [--help] [--version]

Parameters

k= List of field name(s) as unit of division.
O= Create list of sequentially numbered file (serial number starting from 0) in the specified directory.
o= Correspondence table with sequentially numbered file names and values specified as key at k= is output as CSV file.
   Output is printed to standard output if this parameter is not specified.
a= Field name of the path of output specified at o=.
-p Force create directory specified by pathname at O=.

Examples

Example 1: Basic Example

Split the data by corresponding values in item field. Output file names are sequential numbers starting from 0. The key and corresponding number is printed to table.csv.

$ more dat1.csv
item,no
A,1
A,1
A,2
B,1
B,2

$ msep2 k=item O=./output a=fileName o=table.csv i=dat1.csv
#END# kgsep2 O=./output a=fileName i=dat1.csv k=item o=table.csv
$ ls ./output
  0
  1
$ more table.csv
item%0,fileName
A,./output/0
B,./output/1
$ more output/0
item%0,no
A,1
A,1
A,2
$ more output/1
item%0,no
B,1
B,2

Example 2: Multiple key fields

Each file name is created according to the sequential number using item,no as the composite key field. The key field and its corresponding sequential file names are printed to table.csv.

$ more dat1.csv
item,no
A,1
A,1
3.51. **MSEP2 - SEPARATE RECORDS AND RETURN FIELDS TABLE**

A,2
B,1
B,2

$ msep2 k=item,no O=./output2 a=fileName o=table.csv i=dat1.csv

#END# kgsep2 O=./output2 a=fileName i=dat1.csv k=item,no o=table.csv

$ ls ./output2
0
1
2
3

$ more table.csv
item%0,no%1,fileName
A,1,./output2/0
A,2,./output2/1
B,1,./output2/2
B,2,./output2/3

$ more output/0
itemX0,no
A,1
A,1
A,2

**Related Command**

**msep**: Use this command to include the field header name in file name.
3.52 **msetstr - Add String Column**

Add specified string as new field(s) in all rows. More than one fields can be added.

**Format**

```
msetstr v= a= [i=] [o=] [-nfn] [-nfno] [-x] [--help] [--version]
```

**Parameters**

- **v=** List of character strings to add. NULL value is added if the value is not specified.
- **a=** Add field name.
  - The number of string and field name of specified at `v=` must be the same number. `v=`,

**Examples**

**Example 1: Basic Example**

Calculate the date by setting a reference date (defined as January 01, 2007) and add the string `20070101` in all lines and save the output as a new column named `ReferenceDate`.

```
$ more dat1.csv
customer,date
A,20081202
A,20081204
B,20081203
$ msetstr v=20070101 a=ReferenceDate i=dat1.csv o=rsl1.csv
#END# kgsetstr a=ReferenceDate i=dat1.csv o=rsl1.csv v=20070101
$ more rsl1.csv
customer,date,ReferenceDate
A,20081202,20070101
A,20081204,20070101
B,20081203,20070101
```

**Example 2: Add multiple fields**

```
$ msetstr v=20070101,20070201 a=RefDate1,RefDate2 i=dat1.csv o=rsl2.csv
#END# kgsetstr a=RefDate1,RefDate2 i=dat1.csv o=rsl2.csv v=20070101,20070201
$ more rsl2.csv
customer,date,RefDate1,RefDate2
A,20081202,20070101,20070201
A,20081204,20070101,20070201
B,20081203,20070101,20070201
```

**Example 3: Add column with null values**

```
$ msetstr v= a=NewColumn i=dat1.csv o=rsl3.csv
#END# kgsetstr a=NewColumn i=dat1.csv o=rsl3.csv v=
$ more rsl3.csv
customer,date,NewColumn
A,20081202,
A,20081204,
B,20081203,
```

**Related Command**

**mcal**: Use the `if` function to add a fixed string according to differently conditions for each row.
3.53 mshare - Calculate Composition Ratio

Calculate the composition ratio of the fields specified in f=, and add results as a new field.

Format

mshare f= [k=] [i=} [o=} [-tmpPath] [-nfn] [-nfno] [-x] [-q] [--help] [--version]

Parameters

f=  Calculate share value of field(s) (multiple fields can be specified) specified here. Specify the new field name using : (colon). Example: f=Quantity:volume share.

k=  Specify the list of field name(s) (multiple items can be specified) as the unit to calculate share. If the key is omitted, all lines are assumed to have the same key value.

Example

Example 1: Basic Example

Calculate the share of "quantity" and "amount" fields for each "customer". Save the output in columns "volume share" and "share amount".

```bash
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ mshare k=customer f=quantity:qtyShare,amount:amountShare i=dat1.csv o=rsl1.csv
#END# kgshare f=quantity:qtyShare,amount:amountShare i=dat1.csv k=customer o=rsl1.csv
$ more rsl1.csv
customer%0,quantity,amount,qtyShare,amountShare
A,1,10,0.3333333333,0.3333333333
A,2,20,0.6666666667,0.6666666667
B,1,15,0.2,0.3333333333
B,3,10,0.6,0.2222222222
B,1,20,0.2,0.4444444444
```

Related Command
3.54  **msim - Calculate Similarity Between Two Variables**

Find out the degree of similarity between two variable fields (distance) at \( f= \) parameter, specify the degree of similarity (distance) function at \( c= \) parameter to derive the similarity matrix.

**Format**

```
msim c= f= [a=] [k=} [-d] [i=} [o=} [bufcount=] [-nfn] [-nfno] [-x] [-q] [precision=} [--help] [--version]
```

**Parameters**

- **k=** Field(s) (multiple items can be specified) specified here is used as the unit of calculation.
- **f=** Field names for the calculation of degree of similarities between two fields.
- **c=** Specify the similarity measure(s) (distance) (multiple fields can be specified). As shown in the example below, the field name of the similarity measure results can be defined by using a : (colon). If the name of field is not defined with colon, the type of degree of similarity (distance) is used as the field name.
  - Example: `msim f=x,y,z c=pearson:Pearson product-moment correlation coefficient, euclid:Euclidean distance, cosine:Cosine`
- **a=** Specify the field name that indicates the name of the two variables. Specify the two arguments with a comma. Field names `f1d1,f1d2` are used if `a=` is not defined.
- **-d** Output as diagonal matrix and upper triangular matrix. Only the lower triangular matrix of similarity matrix is shown if `-d` option is not specified, but both upper triangular matrix and diagonal matrix are shown by when `-d` option is specified.

**Definition of similarity (distance)**

**Real vector**

Definition of size for the degree of similarity (or distance) in relation to two real number vectors \( \mathbf{x} = (x_1, x_2, \ldots, x_n) \), \( \mathbf{y} = (y_1, y_2, \ldots, y_n) \) is shown in Table 3.24.

**0-1Vector**

Take the value as 0 or 1, the definition of degree of similarity of two 0-1 vectors \( \mathbf{a} = (a_1, a_2, \ldots, a_n) \), \( \mathbf{b} = (b_1, b_2, \ldots, b_n) \) is shown in Table 3.26. The \( f_{jk} \) symbols used in the table, the value of \( a_i, b_i \) is enumerated in different combinations of (0,1), and shown in Table 3.25.

Further, meaning of \( P(\cdot) \) is shown below.

**Examples**

**Example 1: Basic Example**

Calculate the cosine and Pearson's product-moment correlation coefficient for the combination of two items among \( x, y, z \) fields.

```bash
$ more dat1.csv
x,y,z
14,0.17,-14
11,0.2,-1
32,0.16,-2
13,0.33,-2
$ msim c=pearson,cosine f=x,y,z i=dat1.csv o=rsl1.csv
```
### Table 3.24: Summary of degree of similarity for real number vectors

<table>
<thead>
<tr>
<th>Parameter value</th>
<th>Detail</th>
<th>Distance/similarity</th>
<th>Equation definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>covar</td>
<td>Covariance</td>
<td>Degree of similarity</td>
<td>$\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$</td>
</tr>
<tr>
<td>ucovar</td>
<td>Unbiased covariance</td>
<td>Degree of similarity</td>
<td>$\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$</td>
</tr>
<tr>
<td>pearson</td>
<td>Pearson’s product-moment correlation coeff</td>
<td>Degree of similarity</td>
<td>$\frac{\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2} \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \bar{y})^2}}$</td>
</tr>
<tr>
<td>spearman</td>
<td>Spearman’s rank correlation coefficient</td>
<td>Degree of similarity</td>
<td>$\rho_{xy}$</td>
</tr>
<tr>
<td>kendall</td>
<td>Kendall’s rank correlation coefficient</td>
<td>Degree of similarity</td>
<td>$\frac{c}{\sqrt{n(n-1)}}$ Note: $1, 2$</td>
</tr>
<tr>
<td>euclid</td>
<td>Euclidean distance (number)</td>
<td>Distance</td>
<td>$\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$</td>
</tr>
<tr>
<td>cosine</td>
<td>Cosine</td>
<td>Degree of similarity</td>
<td>$\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}}$</td>
</tr>
<tr>
<td>cityblock</td>
<td>City block distance</td>
<td>Distance</td>
<td>$\sum_{i=1}^{n}</td>
</tr>
<tr>
<td>hamming</td>
<td>Hamming distance</td>
<td>Distance</td>
<td>$</td>
</tr>
</tbody>
</table>

Note 1: $c = \{(i, j) \mid (x_i > x_j \text{ and } y_i > y_j) \text{ or } (x_i < x_j \text{ and } y_i < y_j), i > j, i = 1, 2, \ldots, n, j = 1, 2, \ldots, n\}$

Note 2: $d = \{(i, j) \mid (x_i > x_j \text{ and } y_i < y_j) \text{ or } (x_i < x_j \text{ and } y_i > y_j), i > j, i = 1, 2, \ldots, n, j = 1, 2, \ldots, n\}$

### Table 3.25: Combinations of the values of the 2 variables in 2 × 2 contingency table

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>1</th>
<th>0</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_i$</td>
<td>1</td>
<td>0</td>
<td>Total</td>
</tr>
<tr>
<td>$f_{11}$</td>
<td>$f_{10}$</td>
<td>$f_1$</td>
<td></td>
</tr>
<tr>
<td>$f_{01}$</td>
<td>$f_{00}$</td>
<td>$f_0$</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>$f_{1}$</td>
<td>$f_0$</td>
<td>$f$</td>
</tr>
</tbody>
</table>

$P(a) = f_1 / f$.

$P(b) = f_0 / f$.

$P(\hat{a}) = f_0 / f$.

$P(a, b) = f_{11} / f$.

$P(a | b) = f_{11} / f_1$
<table>
<thead>
<tr>
<th>Parameter values</th>
<th>Content</th>
<th>Distance/similarity</th>
<th>Equation</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>chi</td>
<td>Chi-square value</td>
<td>Degree of similarity</td>
<td>$\sum_{i=0}^{1} \sum_{j=0}^{1} \frac{f_{ij}-e_{ij}}{e_{ij}}$ Note 1</td>
<td>$0 \sim \infty$</td>
</tr>
<tr>
<td>phi</td>
<td>Phi coefficient</td>
<td>Degree of similarity</td>
<td>$\frac{f_{11}f_{00} - f_{10}f_{01}}{\sqrt{f_{10}f_{01}f_{00}f_{11}}}$</td>
<td>$-1.0 \sim 1.0$</td>
</tr>
<tr>
<td>jaccard</td>
<td>Jack card factor</td>
<td>Degree of similarity</td>
<td>$\frac{P(a,b)}{P(a)P(b)}$</td>
<td>$0.0 \sim 1.0$</td>
</tr>
<tr>
<td>support</td>
<td>Support</td>
<td>Degree of similarity</td>
<td>$P(a,b)$</td>
<td>$0.0 \sim 1.0$</td>
</tr>
<tr>
<td>lift</td>
<td>Value of lift</td>
<td>Degree of similarity</td>
<td>$\frac{P(a,b)}{P(a)P(b)}$</td>
<td>$0 \sim \infty$</td>
</tr>
<tr>
<td>confMax</td>
<td>Maximum confidence</td>
<td>Degree of similarity</td>
<td>$\max(P(a</td>
<td>b), P(b</td>
</tr>
<tr>
<td>confMin</td>
<td>Minimum confidence</td>
<td>Degree of similarity</td>
<td>$\min((P(a</td>
<td>b), P(b</td>
</tr>
<tr>
<td>yuleQ</td>
<td>Ren correlation coefficient of yule (Q)</td>
<td>Degree of similarity</td>
<td>$\frac{\alpha-1}{\alpha+1}$ $\text{Note 2}$</td>
<td>$-1.0 \sim 1.0$</td>
</tr>
<tr>
<td>yuleY</td>
<td>Ren correlation coefficient of yule (Y)</td>
<td>Degree of similarity</td>
<td>$\frac{\sqrt{\alpha}}{\sqrt{\alpha+1}}$ $\text{Note 2}$</td>
<td>$-1.0 \sim 1.0$</td>
</tr>
<tr>
<td>kappa</td>
<td>kappa</td>
<td>Degree of similarity</td>
<td>$\frac{P(a+b)P(a+b) - P(a</td>
<td>b)P(b</td>
</tr>
<tr>
<td>oddsRatio</td>
<td>oddsRatio</td>
<td>Degree of similarity</td>
<td>$\frac{P(a</td>
<td>b)P(b</td>
</tr>
<tr>
<td>convMax</td>
<td>Maximum conviction</td>
<td>Degree of similarity</td>
<td>$\max(P(a</td>
<td>b), P(b</td>
</tr>
<tr>
<td>convMin</td>
<td>Minimum conviction</td>
<td>Degree of similarity</td>
<td>$\min(P(a</td>
<td>b), P(b</td>
</tr>
</tbody>
</table>

Note 1: $e_{ij} = \frac{f_{i}.f_{j}}{f_{..}}$

Note 2: $\alpha = \frac{f_{11}f_{00}}{f_{10}f_{01}}$

Example 2: Output diagonal matrix, the upper triangular matrix

Calculate the cosine and Pearson’s product-moment correlation coefficient for the combination of two items between x, y, z fields (with d option).

```bash
$ msim c=pearson,cosine f=x,y,z -d i=dat1.csv o=rsl2.csv
$ more rsl2.csv
fld1,fld2,pearson,cosine
x,x,1,1
x,y,-0.5088704666,0.7860308044
x,z,-0.5338153343,0.3311001423
y,x,-0.5088704666,0.7860308044
y,y,1,1
y,z,0.3311001423,-0.5524409416
z,x,-0.5338153343,0.3311001423
z,y,-0.5524409416,0.3311001423
z,z,1,1
```

Example 3: Calculation based on key unit

Calculate using key field as unit.

```bash
$ msim c=pearson,cosine f=x,y,z -d i=dat1.csv o=rsl2.csv
```

```bash
#END# kgsim c=pearson,cosine f=x,y,z i=dat1.csv o=rsl1.csv
$ more rsl1.csv
fld1,fld2,pearson,cosine
x,y,-0.5088704666,0.7860308044
x,z,-0.5338153343,0.3311001423
y,x,-0.5088704666,0.7860308044
y,z,0.3311001423,-0.5524409416
y,z,0.3311001423,-0.5524409416
z,x,-0.5338153343,0.3311001423
z,y,-0.5524409416,0.3311001423
z,z,1,1
```

```bash
$ more rsl2.csv
fld1,fld2,pearson,cosine
x,y,1,1
x,y,-0.5088704666,0.7860308044
x,z,0.1963041929,-0.5338153343
x,z,0.3311001423,-0.5524409416
y,x,-0.5088704666,0.7860308044
y,x,0.1963041929,-0.5338153343
y,z,0.3311001423,-0.5524409416
y,z,0.3311001423,-0.5524409416
z,x,0.1963041929,-0.5338153343
z,y,-0.5524409416,0.3311001423
z,z,1,1
```
Example 4: Specify the type of degree of similarity

Using the data with 01 values, compute the phi coefficient and Hamming distance.

$ more dat3.csv
x,y,z
1,1,0
1,0,1
1,0,1
0,1,1

$ msim c=hamming,phi f=x,y,z i=dat3.csv o=rsl4.csv
#END# kgsim a=variable1,variable2 c=hamming:HammingDist,phi:PhiCoeff f=x,y,z i=dat3.csv o=rsl5.csv
$ more rsl4.csv
variable1,variable2,HammingDist,PhiCoeff
x,y,0.75,-0.5773502692
x,z,0.5,-0.3333333333
y,z,0.75,-0.5773502692

Example 5: Rename the column containing degree of similarity

Using the data with 01 values, compute the phi coefficient and Hamming distance and change the output field name.

$ msim c=hamming:HammingDist,phi:PhiCoeff a=variable1,variable2 f=x,y,z i=dat3.csv o=rsl5.csv
#END# kgsim a=variable1,variable2 c=hamming:HammingDist,phi:PhiCoeff f=x,y,z i=dat3.csv o=rsl5.csv
$ more rsl5.csv
variable1,variable2,HammingDist,PhiCoeff
x,y,0.75,-0.5773502692
x,z,0.5,-0.3333333333
y,z,0.75,-0.5773502692

Related Commands

mstats : Calculate the statistics of one variable.
mmvsim : Calculate sliding window similarity measure.
3.55 **mslide - Slide Data Series**

Shift data series in the specified column to a new column by specified number of times. For example, the function can be used to calculate difference between data items in the same field such as deriving the rate of return using daily stock price data (today’s stock price/previous day’s stock price).

Table 3.27 - 3.30 below highlights a commonly used example.

Table 3.27: input data

<table>
<thead>
<tr>
<th>date</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/6</td>
<td>1</td>
</tr>
<tr>
<td>4/7</td>
<td>2</td>
</tr>
<tr>
<td>4/8</td>
<td>3</td>
</tr>
<tr>
<td>4/9</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.28: f=val:nextVal

<table>
<thead>
<tr>
<th>date</th>
<th>val</th>
<th>nextVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4/7</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4/8</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.29: f=val:nextVal -r

<table>
<thead>
<tr>
<th>date</th>
<th>val</th>
<th>nextVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4/8</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4/9</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3.30: f=val t=2

<table>
<thead>
<tr>
<th>date</th>
<th>val</th>
<th>val1</th>
<th>val2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/6</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4/7</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.27 shows the input data containing daily total values for four consecutive days. The figures could represent the changes in supermarket sales or stock price trends. Calculate the rate of increase between two days (for simplicity, assume “rate of increase = rate of day2/rate of day1”).

The input data contains data series from 4/6 to 4/9. In Table 3.28, the data series is shifted up by one line and stored in a new column (newVal). The rate of increase is calculated from nextVal/val using mcal command. After the records have shifted, the last record in the input data dated 4/9 is not longer available. In this case, specify the -n option to print NULL values.

Table 3.28 shows the data output when sliding up the data series by one row, it is also possible to slide in reverse order by sliding the first row to a new column as shown in (Table 3.29).

Furthermore, t= allows user to specify the number of times to slide up or down. Table3.30 shows the result when t=2. The following function is the same as using mslide consecutively:

"mslide f=val:val1 mslide f=val1:val2—".

When t= parameter is used, field names are created for each new column based on the field name specified at f= followed by an incremental value. When t= parameter is used with -l option, data from the initial column and the last column is displayed.

The functions of mwindow and mslide are similar. The difference between the two is that mslide is used for calculation between data items, and the command is often followed by mcal or msel. mwindow is used for calculation by row, which is usually followed by msum or mavg.

**Format**

```plaintext
mslide f= [s=] [k=key] [t=} [-r] [-n] [-l] [i=} [o=} [-nfn] [-nfn] [-x] [-q] [--help] [--version]
```

- **f=** Field name for sliding records. Multiple fields can be specified. The If you do not specify t=, the field name can be specified by f=fieldname:newfieldname.
- **s=** After the specified field is sorted (multiple fields can be specified), records are shifted. s= parameter is required when -q option is not specified.
- **k=** Specify the field for shifting of records.
- **t=** Number of times (rows) to shift. Default value is t=1 if this parameter is not defined.
- **-r** Shift records in the opposite direction (shift the first record below).
- **-n** Print a null value if next (or previous) line is not available.
- **-l** Print results from the final shift.

**Examples**

**Example 1: Basic Example**

```
$ more dat1.csv
date,val
```

```plaintext
mslide f= [s=] [k=key] [t=} [-r] [-n] [-l] [i=} [o=} [-nfn] [-nfn] [-x] [-q] [--help] [--version]
```
Example 2: Slide rows in reverse direction

```bash
$ mslide s=date f=val:newVal -r i=dat1.csv o=rsl2.csv
#END# kgslide -r f=val:newVal i=dat1.csv o=rsl2.csv s=date
$ more rsl2.csv
date%0,val,newVal
20130407,2,1
20130408,3,2
20130409,4,3
```

Example 3: Slide records more than once

```bash
$ mslide s=date f=val t=2 i=dat1.csv o=rsl3.csv
#END# kgslide f=val i=dat1.csv o=rsl3.csv s=date t=2
$ more rsl3.csv
date%0,val,val1,val2
20130406,1,2,3
20130407,2,3,4
20130408,3,4,5
```

Example 4: Display output from the last column shifted

```bash
$ mslide s=date f=val t=2 -l i=dat1.csv o=rsl4.csv
#END# kgslide -l f=val i=dat1.csv o=rsl4.csv s=date t=2
$ more rsl4.csv
date%0,val,val2
20130406,1,3
20130407,2,4
20130408,3,5
```

Example 5: Change multiple field names

```bash
$ mslide s=date f=date:d_,val:v_ t=2 i=dat1.csv o=rsl5.csv
#END# kgslide f=date:d_,val:v_ i=dat1.csv o=rsl5.csv s=date t=2
$ more rsl5.csv
date%0,val_d_1,d_2,v_1,v_2
20130406,1,20130407,20130408,2,3
20130407,2,20130408,20130409,3,4
```

Related Command
3.56  msortf - Sort Records

Sort records according to the field defined at f= parameter. This command uses quicksort algorithm and it is not a stable sort (original order is retained for rows with same key value).

Format

msortf f=[i=] [o=] [tmpPath=] [-nfn] [-nfno] [-x] [--help] [--version]

Parameters

f=  Specify the column name where record values will be sorted accordingly.
Four types of sequence order can be specified namely numeric, string, ascending, descending.
Specify %n after the field name, followed by n or r.
Character string ascending order: field name (% is not specified), character string descending order: f=field%r,
numeric ascending order: f=field%n, numeric descending order: f=field%nr.

Remarks

1. Character string fields specified at k= may not be sorted correct when %n is specified.
2. When k= is not specified, specify the files in merging order at i= (same as mcat).
3. When key field include NULL values, NULL value is treated as a value less than any value.
4. Field names of all input data specified at i= is assumed to have the same field names, whereas mcat has more flexibility in field names.

Examples

Example 1: Basic example

Sort by item and date.

```
$ more dat1.csv
item,date,quantity,price
A,20081201,10,200
A,20081201,10,100
A,20081201,10,100
A,20081201,10,100
B,20081203,5,50
B,20081201,2,500
B,20081201,3,300

$ msortf f=item,date i=dat1.csv o=rsl1.csv
#END# kgsortf f=item,date i=dat1.csv o=rsl1.csv
$ more rsl1.csv
item,date,quantity,price
A,20081201,10,200
A,20081201,10,100
A,20081201,10,100
A,20081203,5,50
B,20081201,2,500
B,20081201,3,300
```
Example 2: Sort by quantity in descending order and price in ascending order.

```bash
$ msortf f=quantity%nr,price%n i=dat1.csv o=rsl2.csv
#END# kgsortf f=quantity%nr,price%n i=dat1.csv o=rsl2.csv
$ more rsl2.csv
item, date, quantity, price
A, 20081201, 10, 100
A, 20081201, 10, 200
B, 20081203, 5, 50
B, 20081201, 4, 40
A, 20081201, 3, 300
B, 20081201, 2, 500
```

Advanced parameters

- **pways=** Merge multiple files simultaneously ([2-100]: default 32) [Optional]
  Specify number of files to merge at a time while sorting multiple files.

- **blocks=** Number of buffer block ([1-1000]: default 100 1blk=400KB) [Optional]
  Specify memory size limit in the block size when sorting in memory.
  Maximum size for 1 block is 4. Default = 400KB.

- **maxlines=** Row fetch limit of memory sort ([100-10,000,000]: 500,000 defaults) [Optional]
  Specify the maximum number of records sorted at once in memory.
  Set -block limit and -maxlines limit depending on the average size of record in the data.

- **threadCnt=** Number of threads to use when sorting in memory ([1-50] Default: 8) [Optional]
  Specify the number of threads for sorting through multi-threading function.

Notes on sorting order of CSV special characters

`msortf` interprets and sorts CSV special characters (e.g. comma and double quotes) differently than the sort command in UNIX. The data fields/columns are separated by comma character. For example, the values in the first column (f1) from the first row onwards are represented by the following ASCII characters: a - (0x61) null - (0x00) space - (0x20) + (0x2b) - (0x2d) , (0x2c) " (0x22) Comma and double quotes is treated as special characters in CSV is enclosed in double quotes. For ease of illustration, "x" is populated in the second column f2 for all records as follows.

```
<table>
<thead>
<tr>
<th>f1, f2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, x</td>
</tr>
<tr>
<td>, x</td>
</tr>
<tr>
<td>, x</td>
</tr>
<tr>
<td>+, x</td>
</tr>
<tr>
<td>-, x</td>
</tr>
<tr>
<td>&quot;&quot;, &quot;x</td>
</tr>
<tr>
<td>&quot;***&quot;, x</td>
</tr>
</tbody>
</table>
```

The statement "`msortf f=f1`" sorts the data as follows. The sort order for CSV format special characters (null, space, double quotation, +, comma,-, a) is explained.

```
<table>
<thead>
<tr>
<th>f1, f2</th>
</tr>
</thead>
<tbody>
<tr>
<td>, x</td>
</tr>
<tr>
<td>&quot;***&quot;, x</td>
</tr>
<tr>
<td>+, x</td>
</tr>
<tr>
<td>&quot;&quot;, &quot;x</td>
</tr>
<tr>
<td>-, x</td>
</tr>
<tr>
<td>a, x</td>
</tr>
</tbody>
</table>
```
Benchmark Test

The benchmark test described here shows the performance of msort and msortf. The input data consist of 6 fields and all data values are uniform random numbers.

---

<table>
<thead>
<tr>
<th>key, fld1, fld2, fld3, fld4, fldn</th>
</tr>
</thead>
<tbody>
<tr>
<td>95547922, 162, 159, 192, 118, 74</td>
</tr>
<tr>
<td>81438069, 138, 157, 155, 122, 58</td>
</tr>
<tr>
<td>26885062, 129, 199, 133, 198, 75</td>
</tr>
<tr>
<td>32651684, 180, 107, 123, 170, -14</td>
</tr>
<tr>
<td>10245631, 164, 103, 159, 154, -63</td>
</tr>
<tr>
<td>15145156, 182, 191, 175, 107, -60</td>
</tr>
<tr>
<td>29254245, 188, 185, 129, 124, 5</td>
</tr>
<tr>
<td>85423170, 116, 164, 175, 113, 57</td>
</tr>
<tr>
<td>55155879, 105, 163, 195, 167, 25</td>
</tr>
<tr>
<td>66997216, 195, 139, 195, 113, 39</td>
</tr>
</tbody>
</table>

---

Compare number of key types and values

The sample data size is 1 million, the following table shows the results according to variation in types of key values at 2, 10, 100, 1000, 10000. Data in the "random number" column is generated using the maximum limit of the random number as key. Data is sorted according to the values in "random number ascending / descending order" column before the benchmark test. The comparison table shows the processing results of msort, MUSASHI xtsort command, and UNIX sort command against the msortf command. The sort command sort one or more sort keys extracted from each line of input, whereas "sort -k1" sorts data on the first column.

The last 3 rows of the table show the result of msortf, xtsort and sort sorted on numeric value stored in the first key field.

- Unit: seconds. Measurement in real time from beginning to end of program using the time command.
- Environment: iMac, Mac OS X 10.5 Leopard, 2.8GHz Intel Core 2 Duo, 4GB memory

<table>
<thead>
<tr>
<th>No.</th>
<th>Command</th>
<th>2 Types</th>
<th>10 Types</th>
<th>100 Types</th>
<th>1000 Types</th>
<th>10000 Types</th>
<th>Rand</th>
<th>Rand Asc</th>
<th>Rand Desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>msortf f=$key</td>
<td>0.29</td>
<td>0.33</td>
<td>0.37</td>
<td>0.40</td>
<td>0.43</td>
<td>0.50</td>
<td>0.29</td>
<td>0.28</td>
</tr>
<tr>
<td>(2)</td>
<td>xtsort -k key</td>
<td>1.25</td>
<td>1.24</td>
<td>1.22</td>
<td>1.20</td>
<td>1.19</td>
<td>1.12</td>
<td>0.85</td>
<td>1.00</td>
</tr>
<tr>
<td>(3)</td>
<td>sort -k1</td>
<td>16.96</td>
<td>16.63</td>
<td>16.05</td>
<td>15.56</td>
<td>15.08</td>
<td>13.68</td>
<td>6.85</td>
<td>7.13</td>
</tr>
<tr>
<td>(4)</td>
<td>msortf f=key%n</td>
<td>0.46</td>
<td>0.56</td>
<td>0.65</td>
<td>0.72</td>
<td>0.79</td>
<td>1.02</td>
<td>0.59</td>
<td>0.59</td>
</tr>
<tr>
<td>(5)</td>
<td>xtsort -k key%n</td>
<td>2.52</td>
<td>2.72</td>
<td>2.96</td>
<td>3.16</td>
<td>3.21</td>
<td>3.22</td>
<td>2.31</td>
<td>2.32</td>
</tr>
<tr>
<td>(6)</td>
<td>sort -k1 -n</td>
<td>16.65</td>
<td>14.52</td>
<td>11.54</td>
<td>8.56</td>
<td>5.71</td>
<td>0.95</td>
<td>0.33</td>
<td>0.36</td>
</tr>
</tbody>
</table>

msortf is 2 to 5 times faster than xtsort. In relation to sort, it can be more than ten times faster depending on the conditions. This command uses the exactly the same quick sort algorithm as in MUSASHI, however, in MCMD multi-threading is used for the parallel processing of sort in separate threads. The impact of the difference is shown.

Next, the experiment shows the change in speed of character string sorting from 1 million records to 10 million records given the number of key types is set as 100 and the maximum value. The comparison of the two commands msortf and xtsort is shown in Figure 3.5, 3.6.

Related Command
3.56. **MSORTF - SORT RECORDS**

### Figure 3.3: Compare sort results on character strings with msort, msortf, xtsort on various key types. (x-axis: number of the key types, y-axis: seconds)

### Figure 3.4: Compare sort results on numerical values with msortf, xtsort on various key types (x-axis: number of key types, y-axis: seconds)
Figure 3.5: Sorting results with 100 key types (x-axis: number of records, y-axis: seconds)

Figure 3.6: Sorting results with different key types using random number (maximum) (x-axis: number of records, y-axis: seconds)
3.57 mstats - Calculate Statistics of 1 Variable

Specify the numeric fields in the parameter \( f = \) and calculate the statistics specified in the parameter \( c = \). Specify the aggregate key unit at \( k = \). NULL value in the specified field(s) at \( f = \) are ignored. However, if all records include NULL values, NULL values will be included in the output.

**Format**

\[
\text{mstats } c= f= [k=] [i=] [o=] [-nfn] [-nfno] [-x] [-q] [precision=] [--help] [--version]
\]

**Parameters**

\( k = \) Compute aggregate statistics on the key field(s) specified (multiple fields can be specified).
\( f = \) Fields for which statistics are computed (multiple fields can be specified).
\( c = \) Statistics (select one from the list below)

- sum
- mean
- count
- ucount
- devsq
- var
- uvar
- sd
- usd
- USD
- cv
- min
- qtile1
- median
- qtile3
- max
- range
- qrange
- mode
- skew
- uskew
- kurt
- ukurt

**List of statistics**

**Examples**

**Example 1: Basic**

Calculate the statistical sum of "quantity" and "amount" field for each "customer".

```
$ more dat1.csv
customer,quantity,amount
A,1,10
B,5,20
C,1,15
C,3,10
C,1,21

$ mstats k=customer f=quantity,amount c=sum i=dat1.csv o=rsl1.csv
#END# kgstats c=sum f=quantity,amount i=dat1.csv k=customer o=rsl1.csv

$ more rsl1.csv
customer%0,quantity,amount
A,1,10
B,5,30
C,5,46
```

**Example 2: Basic Example 2**

Calculate the statistical maximum value.

```
$ mstats k=customer f=quantity,amount c=max i=dat1.csv o=rsl2.csv
#END# kgstats c=max f=quantity,amount i=dat1.csv k=customer o=rsl2.csv

$ more rsl2.csv
customer%0,quantity,amount
A,1,10
B,5,20
C,3,21
```

**Related Commands**

- `msim`: Find out the bivariate statistics.
- `mavg`: Commands specific to \( c = \text{avg} \).
msum: Commands specific to c=sum.

mcount: Unlike c=count, this count the number of rows for each aggregate key.
### Value of \( c= \) Description

<table>
<thead>
<tr>
<th>Value of ( c= )</th>
<th>Description</th>
<th>Equation</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Count (Except NULL value)</td>
<td>( n: ) Number of non-NULL records</td>
<td>It can not be applied to character string field.</td>
</tr>
<tr>
<td>ucount</td>
<td>Unique count</td>
<td>( u_n: ) Number of duplicate values removed</td>
<td>It can not be applied to character string field.</td>
</tr>
<tr>
<td>sum</td>
<td>Total</td>
<td>( sum = \sum_{i=1}^{n} x_i )</td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>Arithmetic mean</td>
<td>( m = \frac{1}{n} \sum_{i=1}^{n} x_i )</td>
<td></td>
</tr>
<tr>
<td>devsq</td>
<td>Sum of squared deviation</td>
<td>( S = \sum_{i=1}^{n} (x_i - m)^2 )</td>
<td></td>
</tr>
<tr>
<td>var</td>
<td>Variance</td>
<td>( s^2 = \frac{1}{n} S )</td>
<td></td>
</tr>
<tr>
<td>uvar</td>
<td>Variance (unbiased estimate)</td>
<td>( u^2 = \frac{1}{n-1} S )</td>
<td></td>
</tr>
<tr>
<td>sd</td>
<td>Standard deviation</td>
<td>( s = \sqrt{s^2} )</td>
<td></td>
</tr>
<tr>
<td>usd</td>
<td>Standard deviation (unbiased variance)</td>
<td>( u = \sqrt{u^2} )</td>
<td>commonly used standard deviation</td>
</tr>
<tr>
<td>USD</td>
<td>Unbiased standard deviation</td>
<td>Omission</td>
<td>Accurate unbiased estimation</td>
</tr>
<tr>
<td>cv</td>
<td>Coefficient of variation</td>
<td>( cv = s/mx100% )</td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>Mode</td>
<td>( mode: ) Most frequent value</td>
<td></td>
</tr>
</tbody>
</table>

| min             | Minimum value | \( min = \min, x_i \) | Print the value of the smaller value if the frequency is same |
| max             | Maximum value | \( max = \max, x_i \) | Print NULL if values are different. |
| range           | Range        | \( r = max - min \) |       |
| median          | Median       | \( Q2 = Secondquartilewhensortedinascedingorder \) |       |
| qtile1          | First quartile | \( Q1 = Firstquartilewhensortedinascedingorder \) |       |
| qtile3          | Third quartile | \( Q3 = Thirdquartilewhensortedinascedingorder \) |       |
| qrange          | Interquartile range | \( rq = Q3 - Q1 \) |       |
| skew            | Skewness     | \( \frac{1}{n} \sum_{i=1}^{n} (x_i - m)^3 \) |       |
| uskew           | Skewness (unbiased estimate) | omitted |       |
| kurt            | Kurtosis     | \( \frac{1}{n} \sum_{i=1}^{n} (x_i - m)^4 - 3.0 \) |       |
| ukurt           | Kurtosis (unbiased estimated) | omitted |       |
3.58  msum - Sum of Column

 Aggregate the sum of values in the records at the specified field defined at \( f = \) parameter for records with the same key value defined at \( k = \).

 Format

\[
\text{msum } f = [k=] [-n] [i=] [o=] [tmpPath=] [-nfn] [-nfno] [-x] [-q] [precision=] [--help] [--version]
\]

 Parameters

\( k = \) Specify list of field name(s) (multiple fields can be specified) as aggregate unit.
\( f = \) Aggregate the values specified at the field(s) (multiple items can be specified). Records with NULL values are ignored.
\( -n \) If NULL exist in the field defined at \( f = \), the result will return NULL.

 Example

 Example 1: Basic Example

Calculate the total value of "quantity" and "amount" for each "customer". Save the output with field names "total quantity" and "total amount".

\[
\begin{align*}
\text{\$ more dat1.csv} \\
\text{customer,quantity,amount} \\
A,1,10 \\
A,2,20 \\
B,1,15 \\
B,3,10 \\
B,1,20 \\
\text{\$ msum k=customer f=quantity:quantitySum,amount:amountSum i=dat1.csv o=rsl1.csv} \\
\text{#END# kgsum f=quantity:quantitySum,amount:amountSum i=dat1.csv k=customer o=rsl1.csv} \\
\text{\$ more rsl1.csv} \\
\text{customer%0,quantitySum,amountSum} \\
A,3,30 \\
B,5,45 
\end{align*}
\]

 Related Commands

 mhashsum : Compute sum without sorting by aggregate key in advance.
 mavg : Compute average.
 mstats : Compute a variety of statistics.
3.59  msummary Calculate Statistics for 1 Variable

Calculate the type of statistics specified at c= parameter for fields specified at f= parameter.

Format

```
msummary c= f= [a=] [k=] [i=] [o=] [-nfn] [-nfno] [-x] [-q] [precision=] [--help] [--version]
```

Parameters

- **k**: Compute statistics based on the key field(s) specified (multiple fields can be specified).
- **f**: Field lists for computation of statistical summary (multiple fields can be specified).
  - When -x, -nfn option is specified, specify the field number (0).
- **c**: Statistics (multiple fields can be specified)
  - Specify list of statistics delimited by comma.
  - Statistics list:
    - sum/mean/count/ucount/devsq/var/uvar/sd/usd/cv/min/qtile1/qtile3/max/
    - range/qrange/mode/skew/uskew/kurt/ukurt
- **a**: New column name.
- **i**: Results from calculation on column(s) specified at f= parameter (default is nd).

List of Statistics

The list of statistics specified at c= parameter is shown in Table 3.32.

Examples

Example 1: Basic Example

Find out the median and average "quantity" and "amount" by each customer. Save the output in a new column named "type".

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ msummary k=customer f=quantity,amount c=median:medianval,mean:meanval a=type i=dat1.csv o=rsl1.csv
#END# kgsummary a=type c=median:medianval,mean:meanval f=quantity,amount i=dat1.csv k=customer o=rsl1.csv
$ more rsl1.csv
customer%0,type,medianval,meanval
A,quantity,1.5,1.5
A,amount,15,15
B,quantity,1.1,1.666666667
B,amount,15,15
```

Related Commands

- **mstats**: Compute one type of statistics.
Table 3.32: List of Statistics

<table>
<thead>
<tr>
<th>Value of c=</th>
<th>Description</th>
<th>Equation</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Count (Except NULL value)</td>
<td>$n$: Number of non-NULL records</td>
<td>It can not be applied to character string field.</td>
</tr>
<tr>
<td>ucount</td>
<td>Unique count</td>
<td>$un$: Number of duplicate values removed</td>
<td>It can not be applied to character string field.</td>
</tr>
<tr>
<td>sum</td>
<td>Total</td>
<td>$sum = \sum_{i=1}^{n} x_i$</td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>Arithmetic mean</td>
<td>$m = \frac{1}{n} \sum_{i=1}^{n} x_i$</td>
<td></td>
</tr>
<tr>
<td>devsq</td>
<td>Sum of squared deviation</td>
<td>$S = \sum_{i=1}^{n} (x_i - m)^2$</td>
<td></td>
</tr>
<tr>
<td>var</td>
<td>Variance</td>
<td>$s^2 = \frac{1}{n} S$</td>
<td></td>
</tr>
<tr>
<td>uvar</td>
<td>Variance (unbiased estimate)</td>
<td>$u^2 = \frac{1}{n-1} S$</td>
<td></td>
</tr>
<tr>
<td>sd</td>
<td>Standard deviation</td>
<td>$s = \sqrt{s^2}$</td>
<td></td>
</tr>
<tr>
<td>usd</td>
<td>Standard deviation (sort of unbiased variance)</td>
<td>$u = \sqrt{u^2}$</td>
<td>commonly used standard deviation</td>
</tr>
<tr>
<td>cv</td>
<td>Coefficient of variation</td>
<td>$cv = s/m \times 100%$</td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>Mode</td>
<td>$mode$: Most frequent value</td>
<td>Print the value of the smaller value if the frequency is same.</td>
</tr>
<tr>
<td>min</td>
<td>Minimum value</td>
<td>$min = \min_i x_i$</td>
<td></td>
</tr>
<tr>
<td>max</td>
<td>Maximum value</td>
<td>$max = \max_i x_i$</td>
<td></td>
</tr>
<tr>
<td>range</td>
<td>Range</td>
<td>$r = max - min$</td>
<td></td>
</tr>
<tr>
<td>median</td>
<td>Median</td>
<td>$Q_2 = \text{Secondquartilewhensortedinascendingorder}$</td>
<td></td>
</tr>
<tr>
<td>qtile1</td>
<td>First quartile</td>
<td>$Q_1 = \text{Firstquartilewhensortedinascendingorder}$</td>
<td></td>
</tr>
<tr>
<td>qtile3</td>
<td>Third quartile</td>
<td>$Q_3 = \text{Thirdquartilewhensortedinascendingorder}$</td>
<td></td>
</tr>
<tr>
<td>qrange</td>
<td>Interquartile range</td>
<td>$rq = Q_3 - Q_1$</td>
<td></td>
</tr>
<tr>
<td>skew</td>
<td>Skewness</td>
<td>$\frac{\sum_{i=1}^{n} (x_i - m)^3}{s^3}$</td>
<td></td>
</tr>
<tr>
<td>uskew</td>
<td>Skewness (unbiased estimate)</td>
<td>omitted</td>
<td></td>
</tr>
<tr>
<td>kurt</td>
<td>Kurtosis</td>
<td>$\frac{\sum_{i=1}^{n} (x_i - m)^4}{s^4} - 3.0$</td>
<td></td>
</tr>
<tr>
<td>ukurt</td>
<td>Kurtosis (unbiased estimate)</td>
<td>omitted</td>
<td></td>
</tr>
</tbody>
</table>
3.60 mtee - Copy to Multiple Output Files

Contents of input data are copied directly to multiple files and standard output.

Format

```
mtee [o=] [-nostdout] [i=} [-nfn] [-nfno] [-x] [--help] [--version]
```

Parameters

- **o=** Output file names. The contents in the input file is duplicated into multiple files. When this parameter is not defined, mtee copies input to standard output.
- **-nostdout** Copy to output file but not to standard output.

Examples

Example 1: Basic Examples

Copy `dat1.csv` file to two files `rsl1.csv` and `rsl2.csv` In addition, display output on screen through standard output.

```
$ more dat1.csv
  customer,quantity,price
  A,1,10
  A,2,20
  B,1,15
$mtee i=dat1.csv o=rsl1.csv,rsl2.csv
  customer,quantity,price
  A,1,10
  A,2,20
  B,1,15
#END# kgtee i=dat1.csv o=rsl1.csv,rsl2.csv

$ more rsl1.csv
  customer,quantity,price
  A,1,10
  A,2,20
  B,1,15
$ more rsl2.csv
  customer,quantity,price
  A,1,10
  A,2,20
  B,1,15
```

Example 2: Do not print to standard output

When **-nostdout** is specified, the command only copy the two files `rsl1.csv` and `rsl2.csv` but not to standard output.

```
$mtee i=dat1.csv o=rsl1.csv,rsl2.csv -nostdout
#END# kgtee -nostdout i=dat1.csv o=rsl1.csv,rsl2.csv
```

Related Command
CHAPTER 3. COMMAND REFERENCE

3.61 mtonull - Substitute Value for NULL

Specify the target field parameter at f=, substitute the value at the v= parameter with NULL in the data. Select the methods for finding patterns using exact string matching (the default) or substring matching (-sub option).

Format

mtonull f= v= [-sub] [-W] [i=} [o=} [-nfn] [-nfno] [-x] [--help] [--version]

Parameters

f= specify list of field name(s) (multiple fields can be specified) where the values are replaced .

v= Specify the character string value (multiple items can be specified) for matching the field defined at f= parameter. Replace the matched values with NULL.

Options

-sub Rather than matching the exact string, compare character substring with the values in the field specified in the f= parameter and replace the value defined in the v= parameter containing the substring with NULL value.

-W Substring match for wide character strings when the -sub option is specified.

Examples

Example 1: Basic Example

Replace 0 with NULL value in columns quantity and price.

```
$ more dat1.csv
item,quantity,price
A,0,1
B,1,0
C,2,200
D,3,0
E,0,298
$ mtonull f=quantity,price v=0 i=dat1.csv o=rsl1.csv
#END# kgtonull f=quantity,price i=dat1.csv o=rsl1.csv v=0
$ more rsl1.csv
item,quantity,price
A,,1
B,,0
C,2,200
D,,3
E,,298
```

Example 2: Replace a specified number with NULL value

Replace 0 or 1 with NULL value in columns quantity and price.

```
$ mtonull f=quantity,price v=0,1 i=dat1.csv o=rs12.csv
#END# kgtonull f=quantity,price i=dat1.csv o=rs12.csv v=0,1
$ more rs12.csv
item,quantity,price
A,,1
B,,0
C,2,200
D,,3
E,,298
```
Example 3: Substitute substring match

Replace with a NULL value where quantity and price columns contain 0.

```
$ mtonull -sub f=quantity,price v=0 i=dat1.csv o=rsl3.csv
#END# kgtonull -sub f=quantity,price i=dat1.csv o=rsl3.csv v=0
$ more rsl3.csv
item,quantity,price
A,,1
B.,1,
C.2,
D.3,
E,,298
```

Example 4: Substitute character string

Replace the string in the item field that matches character string apple, orange, pineapple with NULL value.

```
$ more dat2.csv
item,price
fruit:apple,100
fruit:peach,250
fruit:grape,300
fruit:pineapple,450
fruit:orange,500
$ mtonull f=item v=apple,orange,pineapple -sub i=dat2.csv o=rsl4.csv
#END# kgtonull -sub f=item i=dat2.csv o=rsl4.csv v=apple,orange,pineapple
$ more rsl4.csv
item,price
,,100
fruit:peach,250
fruit:grape,300
,450
,500
```

Related Command

mnullto: Reversely, replace NULL value with a character string.
3.62 mtra - Convert Vertical Data to Vector

Connect the items from the specified fields in f=, and save the string of items as a vector in a new column (referred to as transaction field). Specify the delimiter character of the items at delim=.

**Format**

```
mtra f= [s=] [k=] [delim=] [-r] [i=] [o=} [-nfn] [-nfno] [-x] [-q] [--help] [--version]
```

**Parameters**

- **f=** Specify field(s) (multiple fields can be specified) where the transaction fields are connected as an item. NULL values will not be included in the vector.
- **k=** Key field name(s) (multiple items can be specified) of the character string pattern.
- **-r** Reverse conversion
- **-r** Reverse conversion
- **delim=** Specify the delimiting character (Space is used as the default delimiter).
- **-r** Reverse conversion
- **-r** Reverse conversion

**Examples**

**Example 1: Basic Example**

Combine the corresponding item for each customer in a string using a space as the delimiter, and save output in the column named transaction.

```
$ more dat1.csv
customer,item
A,a
A,b
B,c
B,d
B,e

$ mtra k=customer f=item:transaction i=dat1.csv o=rsl1.csv
```

```
#END# kgtra f=item:transaction i=dat1.csv k=customer o=rsl1.csv

$ more rsl1.csv
customer%0,transaction
A,a b
B,c d e
```

**Example 2: Use hyphen (-) as item delimiter**

```
$ mtra k=customer f=item:transaction delim=- i=dat1.csv o=rsl2.csv
```

```
#END# kgtra delim=- f=item:transaction i=dat1.csv k=customer o=rsl2.csv

$ more rsl2.csv
customer%0,transaction
A,a-b
B,c-d-e
```

**Example 3: Convert items in descending sort order**

```
$ mtra k=customer s=item%r f=item:transaction i=dat1.csv o=rsl3.csv
```

```
#END# kgtra f=item:transaction i=dat1.csv k=customer o=rsl3.csv s=item%r

$ more rsl3.csv
customer%0,transaction
A,b a
B,e d c
```
Related Commands

\texttt{mvsort} : Vector based transaction data can be processed by a set of commands (with \texttt{mv} as prefix) which handles vector data.

\texttt{mcross} : Rather than converting as transaction data, every item is saved separately as individual field in the output.

\texttt{mtrafd} : Create transaction data using \texttt{field name=value}.

\texttt{mtraflg} : Create transaction data as item with field names.
### 3.63 mtrafld - Convert Transaction Field to Cross (pivot) Table

Create item pairs from the fields specified at \( f \), concatenate the item pairs and save as a new vector field (also referred to as transaction field).

#### Format

```
mtrafld a= \([f=] \[\text{delim=}\] \[\text{delim2=}\] \[\text{-r}\] \[\text{-valOnly}\] \[\text{i=}\] \[\text{o=}\] \[\text{-nfn}\] \[\text{-nfno}\] \[\text{-x}\] \[\text{--help}\] \[\text{--version}\]
```

#### Parameters

- `a=` Specify the transaction field name.
- `f=` List of field name(s) (Multiple fields can be specified) [required when \(-r\) is specified, otherwise, this parameter is optional]
  - The field names specified here will be created as connected items and saved in the transaction field.
  - When \(-r\) option is specified, specify the field name to extract from the transaction data.
  - This parameter is optional when \(-r\) option is specified.
  - If the parameter is not specified, all field names are processed as value pairs.
  - However, when \(f\) is not defined, this command cannot read standard input (using pipe).
- `delim=` Specify the character to separate each transaction field item (default delimiter: space if this parameter is not defined).
- `delim2=` Specify the character to separate value pairs and field name (default character: =).
- `r` Reverse conversion
  - Convert transaction field to cross table.
- `valOnly` When this option is specified, the item do not return the prefix "field name=" in the output.

#### Examples

**Example 1: Basic Example**

Join the fields `price` and `quantity` to a string, rename output field as `transaction`.

```
$ more dat1.csv
customer,price,quantity
A,198,1
B,325,2
C,200,3
D,450,2
E,100,1
$ mtrafld a=transaction f=price,quantity i=dat1.csv o=rsl1.csv
#END# kgtrafld a=transaction f=price,quantity i=dat1.csv o=rsl1.csv
$ more rsl1.csv
customer,transaction
A,price=198 quantity=1
B,price=325 quantity=2
C,price=200 quantity=3
D,price=450 quantity=2
E,price=100 quantity=1
```

**Example 2: Basic Example 2**

Use \(-r\) option to revert the output results back to the original data.

```
$ mtrafld -r a=transaction f=price,quantity i=rsl1.csv o=rsl2.csv
#END# kgtrafld -r a=transaction f=price,quantity i=rsl1.csv o=rsl2.csv
$ more rsl2.csv
customer,price,quantity
A,198,1
B,325,2
C,200,3
```
Example 3: Specify the delimiter

Price and quantity field is separated by \( \_ \_ \_ \) (underscore) character and connected by 1 character string. Use colon and name the output field as transaction.

```bash
$ mtrafld a=transaction f=price,quantity delim=\_ delim2=':' i=dat1.csv o=rsl3.csv
```

Example 4: When data contains NULL value

```bash
$ more dat2.csv
customer,price,quantity
A,198,1
B,,2
C,200,
D,450,2
E,,
```

```bash
$ mtrafld a=transaction f=price,quantity i=dat2.csv o=rsl4.csv
```

Example 5: When data contains NULL value 2

Use -r option to revert the output results back to the original data.

```bash
$ mtrafld -r a=transaction f=price,quantity i=rsl4.csv o=rsl5.csv
```

Example 6: Specify -valOnly option

```bash
$ mtrafld -valOnly f=price,quantity a=transaction i=dat2.csv o=rsl6.csv
```
Related Commands

**mvsort**: Vector based transaction data can be processed by a set of commands (with mv as prefix) which handles vector data.

**mcross**: Rather than converting as transaction data, every item is saved separately as individual field in the output.

**mtra**: Create transaction data using values in the field.

**mtraflg**: Create transaction data with field names.
3.64  mtraflg - Convert Cross (pivot) Table to Transaction Fields

Check whether the field(s) specified in the f= parameter contains NULL value. Fields with non-NULL values are connected as one item and saved as a new vector.

Format

mtraflg a= f= [delim=] [-r] [i=] [o=] [-nfn] [-nfn] [x] [--help] [--version]

Parameters

a=  Specify the transaction field name.

f=  Check the values in the specified field name(s) (multiple fields can be specified) to create transaction data.

   (-r option is specified, extract list of values as the field name of the transaction data)

delim=  Specify the character to separate each transaction field item (Default character is space if this parameter is omitted).

   Character string should not be used. 1 byte character can be specified.

   -r  Reverse conversion

      Convert transaction based data to vertically structured data.

Examples

Example 1: Basic Example

Create a string of vector from the list of non-null values in column egg and milk.

```
$ more dat1.csv
customer,egg,milk
A,1,1
B,,1
C,1,
D,1,1
$ mtraflg f=egg,milk a=transaction i=dat1.csv o=rsl1.csv
#END# kgtraflg a=transaction f=egg,milk i=dat1.csv o=rsl1.csv
$ more rsl1.csv
customer,transaction
A,egg milk
B,milk
C,egg
D,egg milk
```

Example 2: Basic Example 2

Use -r option to revert the output results back to the original data.

```
$ mtraflg -r f=egg,milk a=transaction i=rsl1.csv o=rsl2.csv
#END# kgtraflg -r a=transaction f=egg,milk i=rsl1.csv o=rsl2.csv
$ more rsl2.csv
customer,egg,milk
A,1,1
B,,1
C,1,
D,1,1
```

Example 3: Specify the delimiter

Combine items using the - - (hyphen) as delimiter. Save output in column named transaction.

```
```
```
$ mtraflg f=egg,milk a=transaction delim=-- i=dat1.csv o=rsl3.csv
#END# kgtraflg a=transaction delim=-- f=egg,milk i=dat1.csv o=rsl3.csv
$ more rsl3.csv
customer,transaction
A,egg-milk
B,milk
C,egg
D,egg-milk

Related Commands

mvsort : Vector based transaction data can be processed by a set of commands (with mv as prefix) which handles vector data.
mcross : Rather than converting as transaction data, every item is saved separately as individual field in the output.
mtra : Create transaction data using values in the field.
mtrafl : Create transaction data with the format \{ field name=value \}. 
3.65 muniq - Unique Records

Remove duplicate values and create unique records.

Format

```
muniq [k=]  [i=]  [o=]  [-nfn]  [-nfnc]  [-x]  [-q]  [--help]  [--version]
```

Parameter

```
k=  Specify the field name(s) as the unique identifier of the records.
```

Examples

Example 1: Basic Example

Remove duplicate records in the `date` field.

```
$ more dat1.csv
date,customer
20081201,A
20081202,A
20081202,B
20081202,B
20081203,C

$ muniq k=date i=dat1.csv o=rsl1.csv
#END#

$ muniq k=date,customer i=dat1.csv o=rsl1.csv
#END#

$ more rsl1.csv
date%0,customer
20081201,A
20081202,B
20081203,C
```

Example 2: Delete duplicate rows in multiple columns

Remove duplicate records based on unique values in `date` and `customer` field.

```
$ muniq k=date,customer i=dat1.csv o=rsl2.csv
#END#

$ muniq k=date,customer i=dat1.csv o=rsl2.csv
#END#

$ more rsl2.csv
date%0,customer%1
20081201,A
20081202,B
20081203,C
```

Related Command

```
mbest  :  Use mbest command to select the line number for records with the same key.
```
3.66  mvcat - Combine Vectors

Merge multiple vectors into one vector.

Examples are shown in Table 3.33, 3.34 and 3.35.

<table>
<thead>
<tr>
<th>Table 3.33: Input data in.csv</th>
<th>Table 3.34: Basic example mvcat vf=item1,items2 a=catItems i=in.csv</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>items1,items2</td>
</tr>
<tr>
<td>1</td>
<td>a c,b</td>
</tr>
<tr>
<td>2</td>
<td>a d,a e</td>
</tr>
<tr>
<td>3</td>
<td>b f</td>
</tr>
<tr>
<td>4</td>
<td>e,e</td>
</tr>
</tbody>
</table>

Table 3.35: Retain original vectors before merging

mvcat vf=item1,items2 -A i=in.csv

<table>
<thead>
<tr>
<th>no</th>
<th>items1,items2,new</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a c,b,a b</td>
</tr>
<tr>
<td>2</td>
<td>a d,a e,a d a e</td>
</tr>
<tr>
<td>3</td>
<td>b f,b f</td>
</tr>
<tr>
<td>4</td>
<td>e,e,e</td>
</tr>
</tbody>
</table>

Format

mvcat vf= a= [-A] [i=] [o=} [delim=} [-nfn] [-nfno] [-x] [--help] [--version]

vf= Merge specified field names of vectors (from the input file i=).
    Wildcard can be substituted for a character within the field name.

a= Field name of the merged vector.

-A Add results as a new field. If this option is not specified, the original field (vf=) will be removed.

Examples

Example 1: Merge vectors using wild character

```
$ more dat1.csv
items1,items2,items3,items4
b a c,b,x,y
c c,x,y
e a a,a a a,x,y
$ mvcat vf=items* a=items i=dat1.csv o=rsl1.csv
#END# kgvcat a=items i=dat1.csv o=rsl1.csv vf=items*
$ more rsl1.csv
items
b a c b x y
c c x y
e a a a a a x y
```

Related command
3.67 mvcommon - Select Common Elements of Vector

Within the vector, select common elements specified in reference file.

See examples in Table 3.36–3.39.

<table>
<thead>
<tr>
<th>Table 3.36: Input data</th>
<th>Table 3.37: Reference file</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in.csv</strong></td>
<td><strong>ref.csv</strong></td>
</tr>
<tr>
<td>no</td>
<td>items</td>
</tr>
<tr>
<td>1</td>
<td>a b c</td>
</tr>
<tr>
<td>2</td>
<td>a d</td>
</tr>
<tr>
<td>3</td>
<td>b f e f</td>
</tr>
<tr>
<td>4</td>
<td>f c d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3.38: Basic example</th>
<th>Table 3.39: Selection of unmatched items.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>vf=items m=ref.csv K=item</strong></td>
<td><strong>vf=items m=ref.csv K=item -r</strong></td>
</tr>
<tr>
<td>no</td>
<td>items</td>
</tr>
<tr>
<td>1</td>
<td>a c</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>e</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
</tr>
</tbody>
</table>

Since the mvcommon command reads the whole reference file at once into the memory, note that huge reference file may consume massive amount of memory.

**Format**

mvcommon vf= K= [-r] m=| i=[o=] [delim=] [-nfu] [-nfno] [-x] [--help] [--version]

- **vf=** Specify the field name(s) of vector (from i= input file) for matching.
  - Multiple fields can be specified. Sorting on vectors is not required.
- **m=** Reference file.
- **K=** Join key of reference file (m=) where corresponding taxonomy elements are joined to the vector.
- **-r** Select records where key elements that do not match in vf= and K=.

**Example**

Example 1: Match common elements in multiple vectors

```
$ more dat1.csv
items1,items2
b a c,b b
c c,a d
e a a,a a
$ more ref1.csv
item
a
c
e
$ mvcommon vf=items1,items2 K=item m=ref1.csv i=dat1.csv o=rsl1.csv
#END# kgvcommon K=item i=dat1.csv m=ref1.csv o=rsl1.csv vf=items1,items2
$ more rsl1.csv
items1,items2
a c
c c,a
e a a,a a
```
Example 2: Print output to a new column

Define new column name for `item2` as `new2`.

```
$ mvcommon vf=items1,items2:new2 K=item m=ref1.csv i=dat1.csv o=rsl2.csv
#END# kgvcommon K=item i=dat1.csv m=ref1.csv o=rsl2.csv vf=items1,items2:new2
$ more rsl2.csv
items1,items2,new2
 a c,b b,
c c,a d,a
e a a,a a,a a
```

Related command

`mvjoin`: Join reference element to vector instead of select.
3.68 mvcount - Calculate Vector Size

Calculate the size of vector (number of elements in a vector).

An example is shown in Table 3.40 - 3.41.

### Table 3.40: Input data

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>f</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>f</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 3.41: Basic example

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>f</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format**

```bash
mvcount vf= [i=] [o=] [delim=] [-nfn] [-nfno] [-x] [--help] [--version]
```

**vf** Specify the field names (from input file i=) of vectors to count the number of elements. Field name(s) of result(s) can be defined with : followed after the vector name. Multiple vectors can be specified.

### Example

**Example 1: Count multiple vectors**

```bash
$ more dat1.csv
items1,items2
b a c,b
c c,e a a a a
$ mvcount vf=items1:size1,items2:size2 i=dat1.csv o=rsl1.csv
```

**Related command**
3.69  mvdelim - Change Vector Delimiter

Change delimiter used to separate between string of characters in a vector. However, the delimiter will be removed if an empty string is specified as the delimiter.

Some examples are shown in Table 3.42 - 3.46. When comma is used as delimiter, a pair of double quotation marks is added to the vector (Table 3.44). If a null character is specified as delimiter at v=, the delimiter between characters will be removed (Table 3.45).

Alphabet and chinese characters can be used as delimiter as shown in Table 3.46. Since the character type delimiter is read as character string as part of the vector by other commands such as mvsort, character type delimiter can be specified in delim=.

Table 3.42: input data

<table>
<thead>
<tr>
<th>no</th>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b a a</td>
</tr>
<tr>
<td>2</td>
<td>a a b b</td>
</tr>
<tr>
<td>3</td>
<td>a b b a</td>
</tr>
<tr>
<td>4</td>
<td>a b c</td>
</tr>
</tbody>
</table>

Table 3.43: Basic example : Replace space delimiter with minus character.

<table>
<thead>
<tr>
<th>no</th>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b-a-a</td>
</tr>
<tr>
<td>2</td>
<td>a-a-b-b</td>
</tr>
<tr>
<td>3</td>
<td>a-b-b-a</td>
</tr>
<tr>
<td>4</td>
<td>a-b-c</td>
</tr>
</tbody>
</table>

Table 3.44: Use comma as a delimiter.

<table>
<thead>
<tr>
<th>no</th>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;b,a,a&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;a,a,b,b&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;a,b,b,a&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;a,b,c&quot;</td>
</tr>
</tbody>
</table>

Table 3.45: Remove delimiter between characters in a vector

<table>
<thead>
<tr>
<th>no</th>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>baa</td>
</tr>
<tr>
<td>2</td>
<td>aabb</td>
</tr>
<tr>
<td>3</td>
<td>abba</td>
</tr>
<tr>
<td>4</td>
<td>abc</td>
</tr>
</tbody>
</table>

Table 3.46: Specify more than one character in a vector

<table>
<thead>
<tr>
<th>no</th>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bxxaxxa</td>
</tr>
<tr>
<td>2</td>
<td>axxxbxxbxb</td>
</tr>
<tr>
<td>3</td>
<td>axxbxxbxxaa</td>
</tr>
<tr>
<td>4</td>
<td>axxbxxxe</td>
</tr>
</tbody>
</table>

Format

mvdelim vf= v= [i=] [o=] [delim=] [-nf] [-nfo] [-x] [--help] [--version]

vf=  Field name of vector to replace the delimiter. Multiple fields can be specified.

v=  Define new delimiter. If the parameter is not defined, the delimiter is treated as an empty character.

Example

Example 1: Basic Example

Replace the default space delimiter to _ (underscore).

```bash
$ more dat1.csv
item1
b a c
c c
e a a
$ mvdelim vf=item1 v=_ i=dat1.csv o=rsl1.csv
$ more rsl1.csv
item1
b_a_c
c_c
e_a_a
```
Example 2: Comma

In CSV data with comma delimited characters, when the delimiter of vector is replaced as comma, the entire vector is enclosed in double quotes to differentiate from the delimiter of CSV.

```bash
$ mvdelim vf=item1 v=, i=dat1.csv o=rsl2.csv
#END# kgvdelim i=dat1.csv o=rsl2.csv v=, vf=item1
$ more rsl2.csv
item1
"b,a,c"
"c,c"
"e,a,a"
```

Related command
3.70 mvdelnull - Remove a NULL Element in Vector

Remove all NULL elements in the vector. If NULL element exist in vector, there will be consecutive delimiters of the elements. All vectors shown below contains NULL elements. However, for ease of reading, ‘\n’ is added at the end of each vector. Reading from the top row, the 3rd element, 1st element, 4th element are NULL.

```
a b c \n
a b \n
a b c \n```

**Format**

```
mvdelnull vf= [-A] [delim=] i= [o=} [delim=} [-nf] [-nfo] [-x] [--help] [--version]
```

- **vf=** Specify the field name (from input file i= ) which contains NULL element for removal. Multiple files can be specified.
- **-A** Instead of replacing the specified item, this option adds output as a new field. When -A open is specified, the new field name must be specified after :(colon).

**Examples**

**Example 1: Basic example of removing null characters**

```
$ more dat1.csv
items
b a c
c c
e a a

$ mvdelnull vf=items i=dat1.csv o=rsl1.csv
#END# kgvdelnull i=dat1.csv o=rsl1.csv vf=items

$ more rsl1.csv
items
b a c
c c
e a a
```

**Example 2: Example of using .(dot) as delimiting character**

```
$ more dat2.csv
items
b.a..c
c.c
e.a...b.

$ mvdelnull vf=items delim=. i=dat2.csv o=rsl2.csv
#END# kgvdelnull delim=. i=dat2.csv o=rsl2.csv vf=items

$ more rsl2.csv
items
b.a..c
c.c
e.a..b
```

**Example 3: Add output as an new field by specifying -A**

```
$ mvdelnull vf=items:new -A i=dat1.csv o=rsl3.csv
#END# kgvdelnull -A i=dat1.csv o=rsl3.csv vf=items:new

$ more rsl3.csv
items,new
b a c,b a c
```
3.70. **MVDELNULL - REMOVE A NULL ELEMENT IN VECTOR**

\[
\begin{array}{cccc}
  c & c & c \\
  e & a & b , e & a & b \\
\end{array}
\]

**Related Command**

`mvnullto` : Replace NULL element to any value.
3.71 mvjoin - Join Reference Vector Elements

Join vector elements with corresponding taxonomy elements from reference file with the same key. A vector field is shown in Table 3.47 where the column item includes multiple elements separated by a space delimiter.

Table 3.47 - 3.50 highlights some examples.

<table>
<thead>
<tr>
<th>Table 3.47: Input data in.csv</th>
<th>Table 3.48: Reference file ref.csv</th>
</tr>
</thead>
<tbody>
<tr>
<td>no items</td>
<td>item taxo</td>
</tr>
<tr>
<td>1 a b c</td>
<td>a X</td>
</tr>
<tr>
<td>2 a d</td>
<td>b Y</td>
</tr>
<tr>
<td>3 b f e f</td>
<td>c Z</td>
</tr>
<tr>
<td>4 f c d</td>
<td>e X</td>
</tr>
<tr>
<td></td>
<td>f Z</td>
</tr>
</tbody>
</table>

Table 3.49: Basic example

<table>
<thead>
<tr>
<th>vf=items m=ref.csv K=item f=taxo</th>
</tr>
</thead>
<tbody>
<tr>
<td>no items</td>
</tr>
<tr>
<td>1 a b c X Y Z</td>
</tr>
<tr>
<td>2 a d X</td>
</tr>
<tr>
<td>3 b f e f Y Z X X Z</td>
</tr>
<tr>
<td>4 f c d Z Z</td>
</tr>
</tbody>
</table>

Table 3.50: An example defining unmatched taxonomy elements

<table>
<thead>
<tr>
<th>vf=items m=ref.csv K=item f=taxo n=*</th>
</tr>
</thead>
<tbody>
<tr>
<td>no items</td>
</tr>
<tr>
<td>1 a b c X Y Z</td>
</tr>
<tr>
<td>2 a d X</td>
</tr>
<tr>
<td>3 b f e f Y Z X X Z</td>
</tr>
<tr>
<td>4 f c d Z Z</td>
</tr>
</tbody>
</table>

Take note that the mvjoin command read the whole reference file at once into memory, thus huge reference files may consume massive amounts of memory.

Format

mvjoin vf= K= [n=] m= i= [o=] [delim=] [-nfn] [-nfno] [x] [--help] [--version]

vf=     Field name of vector (from i= input file) for joining.
  Multiple fields can be specified. Sorting of the vectors is not required.
m=     Reference file.
K=     Specify key field in reference file (m=) where corresponding taxonomy elements are joined to the vector.
  The sequence of vector should be unique, sorting is not required.
  The output may differ if the string sequence is not unique.
f=     Field name of vector (element) for joining.
n=     Specify the replacement character when the key elements do not match in vf= and K=.
  The vector (element) will not be joined with the reference file when this option not specified.

Example

Example 1: Combine vector with elements from reference file

```
$ more dat1.csv
items
b a c
c c
e a a
$ more ref1.csv
item,taxo
a,X Y
b,X
c,Z Z
$ mvjoin vf=items m=ref1.csv f=taxo i=dat1.csv o=rsl1.csv
$ more dat1.csv
items
b a c
c c
e a a
$ mvjoin vf=items m=ref1.csv f=taxo i=dat1.csv o=rsl1.csv
#END# kgVjoin K=item f=taxo i=dat1.csv m=ref1.csv o=rsl1.csv vf=items
$ more rsl1.csv
```
3.71. MVJOIN - JOIN REFERENCE VECTOR ELEMENTS

Example 2: Join elements to multiple fields

```
$ more dat2.csv
items1,items2
b a c,b b
c c,a d
e a a,a a
$ more ref2.csv
item,taxo
a,X
b,X
c,Y
d,Y
$ mvjoin vf=items1,items2 K=item m=ref2.csv f=taxo i=dat2.csv o=rsl2.csv
$ more rsl2.csv
items1,items2
b a c X X Y,b b X X
c c Y Y,a d X Y
e a a X X,a a X X
```

related command

mvcommon : Use this command to select common elements of vector.
3.72 mvnullto - Replace NULL in vector elements

Replace NULL elements in the vector with an ad hoc value. If NULL element exist in the vector, there will be consecutive delimiters of the elements. All the vectors described below contain NULL element. However, for ease of reading, ‘\n’ is added at the end of each vector. Reading from the top row, the 3rd element, 1st element, and 4th element are NULL.

```
a b c
a b\n
a b c \n```

**Format**

```
mvnullto vf= [v=|-p] [O=] [-A] [delim=] i= [o=} [delim=} [-nfn] [-nfno] [-x] [--help] [--version]
```

- **vf=** Specify the field name (i=on file) to replace NULL.
  - Multiple items can be specified.
- **v=** Specify the replacement string.
- **-p** Replace NULL with the previous element. It can not be specified with v=.
- **O=** Replace all non-NULL elements with the string specified here.
  - Non-NULL value will not be replaced unless specified.
- **-A** Instead of replacing the specified item, this option adds output as a new field.
  - When -A open is specified, the new field name must be specified after :(colon).
  - Example: f=quantity:substitution field name

**Examples**

**Example 1: Replace null characters to the character string ‘null’**

```
$ more dat1.csv
items
b a c
c c
e a b
$ mvnullto vf=items v=null i=dat1.csv o=rsl1.csv
#END# kgvnullto i=dat1.csv o=rsl1.csv v=null vf=items
$ more rsl1.csv
items
b a null c
null c c
e a null null b null
```

**Example 2: Use .(dot) as delimiting character**

```
$ more dat2.csv
items
b a..c
..c.c
e.a..b.
$ mvnullto vf=items v=null delim=. i=dat2.csv o=rsl2.csv
#END# kgvnullto delim=. i=dat2.csv o=rsl2.csv v=null vf=items
$ more rsl2.csv
items
b a.null.c
null.c.c
e.a.null.null.b.null
```

**Example 3: Replace null with the previous value**

```
$ mvnullto vf=items -p i=dat1.csv o=rsl3.csv
#END# kgvnullto -p i=dat1.csv o=rsl3.csv vf=items
```
Example 4: Replace all values except null by specifying O=

Related command

mvdelnull : Delete a NULL element.
3.73 mvreplace - Replace an Element in Vector

Replace vector data with corresponding taxonomy character in the reference file joined by key. Table 3.51 shows items in a vector with sequential elements separated by a space.

The examples are highlighted in Table 3.51 - 3.54.

<table>
<thead>
<tr>
<th>Table 3.51: Input data</th>
<th>Table 3.52: Reference file</th>
</tr>
</thead>
<tbody>
<tr>
<td>in.csv</td>
<td>ref.csv</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>no</td>
<td>items</td>
</tr>
<tr>
<td>1</td>
<td>a b c</td>
</tr>
<tr>
<td>2</td>
<td>a d</td>
</tr>
<tr>
<td>3</td>
<td>b f e f</td>
</tr>
<tr>
<td>4</td>
<td>f c d</td>
</tr>
</tbody>
</table>

Table 3.53: Basic example

<table>
<thead>
<tr>
<th>vf=items m=ref.csv K=item f=taxo</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.54: Define unmatched elements

<table>
<thead>
<tr>
<th>vf=items m=ref.csv K=item f=taxo n=*</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Since the mvreplace command reads the whole reference file at once in memory, note that huge reference file may consume massive amount of memory.

Format

mvreplace vf= K= [n=] m=| i= [o=] [delim=] [-nfn] [-nfno] [-x] [--help] [--version]

vf= Specify the field name of vector (from input file i=).

m= Reference file

K= Key field in reference file (m=) where corresponding taxonomy elements are joined with vector items.
The sequence of the vector is unique, sorting is not required.
The output may differ if the string sequence is not unique.

f= Field name of vector for joining.

n= Specify the replacement character when the elements that do not match in vf= and K=.
The element will not be joined with the reference file when this option not specified.

Example

Example 1: Replace elements in a vector

```
$ more dat1.csv
items
b a c
c c
e a a

$ more ref1.csv
item,taxo
a,X Y
b,X
c,Z Z

$ mvreplace vf=items K=item m=ref1.csv f=taxo i=dat1.csv o=rsl1.csv

#END# kgvreplace K=item f=taxo i=dat1.csv m=ref1.csv o=rsl1.csv vf=items

$ more rsl1.csv
items
X X Y Z Z
```
Example 2: Replace character in multiple elements

```
$ more dat2.csv
items1,items2
b a, c b
b c, a d
e a, a a
$ more ref2.csv
item, taxo
a, X
b, X
c, Y
d, Y
$ mvreplace vf=items1,items2 K=item m=ref2.csv f=taxo i=dat2.csv o=rsl2.csv
#END# kgvreplace K=item f=taxo i=dat2.csv m=ref2.csv o=rsl2.csv vf=items1,items2
$ more rsl2.csv
items1,items2
X X, Y, X
Y Y, X Y
e X, X X
```

Related Command

`mvjoin`: Use `mvjoin` to combine the elements instead of replacing elements.
3.74 mvsort - Sort Vectors

This command sorts series of vectors in column. The vector in the fields shown in Table 3.55 shows multiple character strings separated by space delimiter. Table 3.55 - 3.58 highlight examples sorting vectors.

In Table 3.56, character strings are arranged in ascending order by default. The character strings can be sorted in numerical ascending order by attaching % after the item name followed by n (see Table 3.57), and in reverse order by specifying r after an item name (Table 3.58).

<table>
<thead>
<tr>
<th>Table 3.55: Input data</th>
<th>Table 3.56: Basic usage: Sort vector elements in ascending order.</th>
<th>Table 3.57: Sort numerical in ascending order.</th>
<th>Table 3.58: Sort in numerical descending order.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in.csv</td>
<td>vf=items</td>
<td>vf=items%n</td>
<td>vf=items%nr</td>
</tr>
<tr>
<td>no_Items</td>
<td>no_Items</td>
<td>no_Items%</td>
<td>no_Items%</td>
</tr>
<tr>
<td>1 2 1 13</td>
<td>1 1 13 2</td>
<td>2 2 4 5 5</td>
<td>2 5 5 4 2</td>
</tr>
<tr>
<td>2 4 5 2 5</td>
<td>2 4 5 5</td>
<td>3 14 112</td>
<td>3 112 14</td>
</tr>
<tr>
<td>3 112 14</td>
<td>3 112 14</td>
<td>4 5 31</td>
<td>4 31 5</td>
</tr>
<tr>
<td>4 5 31</td>
<td>4 5 31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Format

mvsort vf= [i=] [o=] [delim=] [-nfn] [-nfno] [-x] [--help] [--version]

vf= Specify the field name(s) of vectors for sorting. Multiple fields can be specified.
    Add n after % after field name to sort in ascending numerical order.
    Add r after % after field name to sort in reverse order.
    Add both n and r to sort in descending numerical order.

Example

Example 1: Sort multiple vectors

Sort item1 data series in ascending order and item2 in numerical ascending order.

```bash
$ more dat1.csv
items1,items2
b a c,10 2
a d,2 5 3
e a a,1
$ mvsort vf=items1%r,items2%n i=dat1.csv o=rsl1.csv
#END# kgvsort i=dat1.csv o=rsl1.csv vf=items1%r,items2%n
$ more rsl1.csv
items1,items2
c b a,2 10
c c,2 3 5
e a a,1
```

Related Command
3.75  **mvuniq - Unique Vector Elements**

This command merges duplicate elements in a vector. However, since the merging process uses a tree structure, the sequence of elements in the output maybe not be in order.

When the `-n` option is specified, the command reads the vector as a sequential series. The vector series is scanned from the beginning of the string, and output unique character strings in the vector.

The examples are highlighted in Table 3.60, 3.61. Table 3.60 shows all merged elements in the data series. When the `-n` option is specified, same elements next to each other are merged in sequential order. In Table 3.61, consecutive elements of b are merged.

<table>
<thead>
<tr>
<th>Table 3.59: Input data</th>
<th>Table 3.60: Basic example</th>
<th>Table 3.61: Merge same elements adjacent to each other in a vector series</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>in.csv</em></td>
<td><em>vf=items i=in.csv</em></td>
<td><em>vf=items -n i=in.csv</em></td>
</tr>
<tr>
<td>no items</td>
<td>no items</td>
<td>no items</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b a a</td>
<td>a b</td>
<td>b a</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a a b b b b</td>
<td>a b</td>
<td>a b</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>a b b a</td>
<td>a b</td>
<td>a b a</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>a b c</td>
<td>a b c</td>
<td>a b c</td>
</tr>
</tbody>
</table>

**Format**

`mvuniq vf= [-n] [i=] [o=] [delim=] [-nfn] [-nfno] [-x] [--help] [--version]`

- `vf=` Specify the target field name(s) of vectors.
- Multiple field name(s) of vectors can be specified.
- `-n` Merges same elements adjacent to each other in the vector.

**Example**

**Example 1: Merges vector elements in multiple fields**

```bash
$ more dat1.csv
items1,items2
b a c,1 1
c c,2 2 3
e a a,3 1
$ mvuniq vf=items1,items2 i=dat1.csv o=rsl1.csv
#EN# kgvuniq i=dat1.csv o=rsl1.csv vf=items1,items2
$ more rsl1.csv
items1,items2
a b c,1
b b b,2 5
a e,1 3
```

**Related Command**
CHAPTER 3. COMMAND REFERENCE

3.76 mwindow - Generate Sliding Window

Replicate original records and shift specified fields. A fixed window with constant width is set when calculating moving averages for time series data. The first element of moving average is obtained by taking the average of the initial fixed subset of the number series. The subset is shifted forward and included the next number following the original subset in the series. This method is known as sliding window calculation.

An example is shown from Table 3.64.

<table>
<thead>
<tr>
<th>Table 3.62: input data</th>
<th>Table 3.63: wk=date:win t=2</th>
<th>Table 3.64: wk=date:win t=2 -r</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>val</td>
<td>win</td>
</tr>
<tr>
<td>4/6</td>
<td>1</td>
<td>4/7</td>
</tr>
<tr>
<td>4/7</td>
<td>2</td>
<td>4/8</td>
</tr>
<tr>
<td>4/8</td>
<td>3</td>
<td>4/8</td>
</tr>
</tbody>
</table>

Table 3.62 shows the input data which contains total daily values of four consecutive days. The figures could represent the changes in sales in supermarket and stock price trends.

This example calculates the moving average from 4/6 to 4/9 with a subset size of 2 for each window. Three window intervals [(4/6,1),(4/7,2)], [(4/7,2),(4/8,3)], [(4/8,3),(4/9,4)] are generated, where [ ] indicates a window, and ( ) indicates a line.

Based on the unique key of each windows (referred as "window key"), the output prints the maximum value of the window (the last row of item can be specified by wk= parameter) and the field name based (Table 3.63). -r option is used as the minimum value (first row of data) of each window (Table 3.64). Afterwards, the output results (Table 3.63) is followed by using mavg to calculate the averages of the data series.

The mmvavg command is equivalent to processing the data with mwindow+mavg as described above. However, mmvavg is 3.5 times faster when experimented with a data set of 200MB for 10 million records with a subset size of 10 for each window.

Format


wk= Specify an unique value from the field name in the input data that identifies the window. After the specified field is sorted, the sliding window is created. %r is added for descending sort order, %n is added for numeric sorting. When sorting in numeric descending order, %nr is added. It is necessary to define the field name of window key after a colon. Multiple fields can be specified.

t= Specify the window size (number of rows).

k= Specify the unit for the generation of windows.

-r Use the first row of data as baseline of sliding window. By default, the last row of data is used as baseline.

-n Print all window intervals even though the window size less than the defined parameter at t=.

i= Input file name

-nfn Input data without field header in the first row.

Example

Example 1: Basic Example

```
$ more dat1.csv
date,val
20130406,1
20130406,2
20130408,3
20130409,4
```
Example 2: Use first row as baseline data

```
$ mwindow wk=date:win t=3 -r i=dat1.csv o=rsl2.csv
#END# kgwindow -r i=dat1.csv o=rsl2.csv t=3 wk=date:win
$ more rsl2.csv
```

```
win%0,date,val
20130406,20130406,1
20130406,20130407,2
20130406,20130408,3
20130407,20130407,2
20130407,20130408,3
20130407,20130409,4
```

Example 3: Print all window intervals even if the window size is less than the defined parameter

```
$ mwindow wk=date:win t=3 -r -n i=dat1.csv o=rsl3.csv
#END# kgwindow -n -r i=dat1.csv o=rsl3.csv t=3 wk=date:win
$ more rsl3.csv
```

```
win%0,date,val
20130406,20130406,1
20130406,20130407,2
20130406,20130408,3
20130407,20130407,2
20130407,20130408,3
20130407,20130409,4
20130408,20130408,3
20130408,20130409,4
20130409,20130409,4
```

Example 4: Example of specifying key field

```
$ more dat2.csv
store,date,val
a,20130406,1
a,20130407,2
a,20130408,3
a,20130409,4
b,20130406,11
b,20130407,12
b,20130408,13
b,20130409,14
```

```
$ mwindow k=store wk=date:win t=2 i=dat2.csv o=rsl4.csv
#END# kgwindow i=dat2.csv k=store o=rsl4.csv t=2 wk=date:win
$ more rsl4.csv
```

```
win%1,store%0,date,val
20130407,a,20130406,1
20130407,a,20130407,2
20130407,a,20130408,3
20130407,a,20130409,4
20130407,b,20130406,11
20130407,b,20130407,12
20130407,b,20130408,13
20130407,b,20130409,14
```

```
Example 2: Use first row as baseline data
```
Example 5: Find out the moving averages between current day and previous day

In the above example, moving average is calculated based on the last day of the window. `mslide` can be used for instances to calculate the moving averages of current day and previous day. The example is as follows:

```bash
$ mslide f=date:date2 -q i=dat1.csv o=rsl5.csv
#END# kgslide -q f=date:date2 i=dat1.csv o=rsl5.csv
$ more rsl5.csv
date,val,date2
20130406,1,20130407
20130407,2,20130408
20130408,3,20130409
```

Example 6: Find out the moving averages from the previous day

```bash
$ mwindow wk=date2:win t=2 i=rsl5.csv o=rsl6.csv
#END# kgwindow i=rsl5.csv o=rsl6.csv t=2 wk=date2:win
$ more rsl6.csv
win%0,date,val,date2
20130408,20130406,1,20130407
20130408,20130407,2,20130408
20130409,20130407,2,20130408
20130409,20130408,3,20130409
```

Related command

`mmvavg` : Command that specializes in computing average of sliding windows.

`mmvstats` : Compute various statistics of sliding windows.
3.77 mxml2csv - Convert XML to CSV

Convert XML formatted data to CSV. The basic rule of conversion is by specifying the element as unit of each record (XML tag) and the element corresponding to the column (or attribute) at the parameters \( k= \), \( f= \). The value of the column can be specified in four ways: text bounded by elements, presence of elements, the value of the attribute, presence of attributes.

When SAX is used as the parser of XML, there is no size constraints of XML. If other encoding besides UTF-8 is used in the XML file, the XML file is converted to UTF-8 and output as CSV. XML data should be structured in a complete, well-formed XML document. Otherwise, the program may return unexpected processing results.

Table 3.65 shows a typical format of XML data. More details are explained in the next section, however, a brief outline is illustrated as follows.

Table 3.66 shows the returned output. Element \(<b>\) is used as the key unit of each record (the element is referred to as "key element"). The column is defined by the attribute of element \( b \) for the value of \( att \) (CSV column name \( b\_att \)). The attributes of element \( c \) includes the value of \( p \) (\( b\_p \)) and flag (\( b\_p\_f \)), as well as the text inside element \( d \) and \( a \) (\( d \), \( a \)).

Here, the flag indicates the presence of specified elements or attributes by the value 0-1 in the output. The text output of the element includes the concatenation of all strings in the range within specified element. However, note that the spaces and the control characters are not included in the output.

**Specifying the key element**

Specify the key element as the key unit of each record (specified in the parameter \( k= \)) with an absolute path. The absolute path is defined from the root directory starting from the symbol ('/'), the hierarchy of elements is separated by the sign '/' . The role of the key elements in this command is to perform the following two functions corresponding to the end tag of the key elements.

- Output one row of column data. In the example above, the end tag of the key elements \(<b>\) has appeared three times, one row of CSV data is inserted as a new line for every instance in the output.
- Initialize the column data. However, it does not initialize the column elements outside the key elements in the output data. In Table 3.66, the text in the output of element \( a \) will be consolidated, even when the end tag of the key elements has emerged, the element \( /a \) is outside the key elements \( /a/b \). As a result, the output data is not initialized.

**Specifying the elements in output column**

If the element defined at \( f= \) is returned as a CSV field in the output, follow the format shown below.

**Element path[\%flag]: CSV field name**
"Field name" is the column name in the CSV output which must be specified.

There are two methods to display elements as columns in the output. The first method is to return the text enclosed by the opening and closing tags of the specified element. The other method is to return 0-1 value to indicate whether the specified element exists. The target element path is defined in the former method, and the flag `%f` is added when using the latter method.

The two methods of specifying the element path include absolute and relative paths. A relative path can be specified by defining the path from elements at `k=`. Table 3.65 shows examples on how to specify the element paths of the XML data.

- Given `k=/a/b`, when `f=:B` is specified, the key elements is the same when relative path is nil. B is the column name of CSV.
- Given `k=/a/b`, `f=c:C` and `f=/a/b/c:C` performs the same function. The former is specified by relative path, while the latter by an absolute path. The CSV field name is defined as `C` for both methods.
- `f=d:D` returns the text within the element `d`. `f=d%f:D` returns output when element `\verb|d|` exists. The field name of CSV is `D`.
- When `k=/a/b`, it is assumed that `f=/a:A`, the column element is outside the key element, the text contained in the element `a` are combined in sequential order. The end tag of the key element exists, however, the end tag of the field element does not exists, this is because data is not cleared at the time.

Specifying attributes in output column

If the attribute defined at `f=` is returned as a CSV column, use the format shown below.

**Element path@Element name[%flag]:CSV field name**

"Field name" is the column name in the CSV output which must be specified.

The method of specifying the element path is the same as specifying the elements in the output field. The attribute name is specified after the element path connected with `@`. By adding `%f` after the element name, the presence of the element can be indicated by 0-1 value in the output.

**Format**

mxml2csv k= f= [i=} [o=} [-nfn] [-nfno] [--help] [--version]

**Parameter**

- `k=` Specify the pathname from the root based on the element as the unit of records.
  - The path starts from the root symbol `/`, and the specified elements are connected with `/`.
  - Example: `/article/sentence/chunk`
- `f=` Specify the element or attribute as fields in the output by delimiting the field names with comma.
  - Format is as follows.
    - **Element path[%flag]:CSV field name**
    - **Element path@Element name[%flag]:CSV field name**
- `i=` Specify the file name of XML data. The input is read from standard input by default when the input file is not specified.

**Examples**

**Example 1: Basic example**

The example below is illustrated in the summary above. Output the 5 CSV fields with `/a/b` set as the key elements.
Example 2: Absolute path

Specification of same element as in the basic example with an absolute path. Output the 5 CSV fields with /a/b as the key elements.

Example 3: Changing key elements

Example of changing a key element to a using an absolute path. Since there is only one end tag a, one row of record will be returned as output. /a/b@att specified at f= appeared twice, the last value is returned as output.

Related command
Chapter 4

mcal

mcal command is developed for computation between columns.

Commands other than mcal carry out record based processing, however, mcal command specializes in item-based calculations. There are more than 100 functions/operators in mcal, user has the flexibility to define various processing functions that combine multiple functions and operators.
4.1 mcal - Computation Between Columns

Define the computation formula at the `c=` parameter, and name the new data attribute at the `a=` parameter. The output of `mcal` is limited to 1 result without exception to simplify the program. For details of the calculation formula, please refer to the section on "Components in the expression".

Format

```
mcal a= c= [i=] [o=] [-nfn] [-nfno] [-x] [precision=] [--help] [--version]
```

Parameters

- `a=` Specify the new column name to store the calculated field.
- `c=` Define an expression with a combination of calculation functions available.

Examples

Basic usage of `mcal` is illustrated in the following example. For more information on the explanation and usage of individual functions and operators, please refer to the corresponding reference.

```
# Input data (dat1.csv)
customer,quantity,unitprice
A,3,10
B,1,15
C,2,20

$ mcal c='${quantity}*${unitprice}' a=amount i=dat1.csv
customer,unitprice,amount
A,3,10,30
B,1,15,15
C,2,20,40

$ mcal c='${quantity}*${unitprice}<=30' a=amountbelow30 i=dat1.csv
customer,unitprice,amountbelow30
A,3,10,1
B,1,15,1
C,2,20,0

$ mcal c='if(top(),${unitprice},#{}+${unitprice})' a=AccumUnitprice i=dat1.csv
customer,unitprice,AccumUnitprice
A,3,10
B,1,15
C,2,20

Considerations when using shell

When using BASH shell on UNIX operating systems, the symbol of operators often have special meaning to the shell. For example, a shell variable is represented by the `$` symbol followed by a character string. On the other hand, mcmd uses the `$` symbol to refers to the value of the data field. Thus, mcmd variables is enclosed in squiggly brackets preceded by the `$` symbol in order not to be misinterpreted as shell variables.

```
$ mcal c='$\{date\}-10'
```

Error message

```
#ERROR# unknown function or operator
```

This error message appears when there is an error in the specified operator or function. For instance, refer to the error message from the concatenation of strings function `cat`.

```
$ mcal c='cat("-",1,2)'
ERROR : unknown function or operator: cat_SNN(cat_SN) (kgcal)
```
The string before underscore character in "cat_SNN" indicates the function name `cat`, subsequently, SNN refers to the type of the argument. S refers to string type, N refers to number type, D refers to date type, and T is the time type, B refers to boolean type. The 3 characters (SNN) specifies 3 arguments. Thus, this error message means 3 arguments SNN of cat function is not registered. The second and third argument is converted to string as follows.

```bash
$ mcal c='cat("-","1","2")'
```

The above returns an error message with 2 characters in parenthesis (SN), this refers to specification of variable or number at the first two parameters, however, only 1 variable is expressed correctly in the function.

### Related Commands

`msel` : Use this command to select the row from the computation result.

## 4.2 Components in the expression

The four key components in `mcal` includes constants, item value, operator, and function. In every component, it is important for user to understand the application of data type. `mcal` handles CSV text data, all values are expressed as character strings, user can define the data type to be handled in mcal. There are five string types in mcal, they are string type (`str`), numeric type (`num`), date type (`date`), time type (`time`), boolean type (`bool`). The following sessions illustrates how each component constitutes to the expression and how to treat the data types.

### 4.3 Constant

<table>
<thead>
<tr>
<th>Data type</th>
<th>Format</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical Value(<code>num</code>)</td>
<td>integer, real number string</td>
<td>Double precision floating point numbers is used internally</td>
<td><code>20, 0.55, 1.5*e10</code></td>
</tr>
<tr>
<td>Character string(<code>str</code>)</td>
<td>&quot;Character string&quot;</td>
<td>Character string enclosed in double quotes</td>
<td>&quot;abc&quot; &quot;日本語&quot;</td>
</tr>
<tr>
<td>Date (<code>date</code>)</td>
<td>0dyyyymmdd</td>
<td>Add &quot;0d&quot; before fixed length year</td>
<td>0d20080923</td>
</tr>
<tr>
<td>Time (<code>time</code>)</td>
<td>0tyyymmmddHHMMSS</td>
<td>Add &quot;0t&quot; before fixed length year, month, date, time, minute and second</td>
<td>0t20080923121115</td>
</tr>
<tr>
<td>Boolean (<code>bool</code>)</td>
<td>0b1, 0b0</td>
<td>Add &quot;0b&quot; before &quot;1&quot; (true) and &quot;0&quot; (false)</td>
<td>0b1, 0b0</td>
</tr>
</tbody>
</table>

### 4.4 Field value

Table 4.2 shows the different data formats in the data field, the type of CSV data varies depends on how it is used and defined by the user.

### 4.5 Wildcard

Wildcard can be used in field names. For example, when using the sum function to compute the total across multiple fields with numeric labels, wildcard can be use to simplify the listing of all fields as one label. For example,
if there are three columns are named A1, A2, A3 in the input data, the total sum of A1, A2, A3 can be calculated as \( \text{sum}(A*) \). It is also possible to specify multiple wildcard such as the expression \( \text{sum}(A*, B*) \).

### 4.6 Value from Previous Row

Use the \# symbol instead of \$ to refer to values from a field in the previous row. However, the function will return null when used on the first record since there is no record preceding the first record. The specification of each data type is shown in 4.3 below.

### 4.7 Values from Next Row

Use the expression to obtain value from the previous record without specifying the field name to obtain the value in the next record. The specification of the data types are shown in 4.4.

It is possible to calculate total by combining if function with top() function. The cumulative calculation on the amount field is shown below.

```sql
$ mcal c='if(top(),${amount},${amount}+#{})' a=cumulativeAmount
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Format</th>
<th>Content of CSV Data</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical value</td>
<td>${fieldname}</td>
<td>Integer, real number (including floating Numerical string)</td>
<td>${amount}, ${stockprice}</td>
</tr>
<tr>
<td>Character string</td>
<td>$s{fieldname}</td>
<td>Character string</td>
<td>$s{gender}, $s{gender}</td>
</tr>
<tr>
<td>Date</td>
<td>$d{fieldname}</td>
<td>Fixed length year month day (yyyymmdd)</td>
<td>$d{date}, $d{orderdate}</td>
</tr>
<tr>
<td>Time</td>
<td>$t{fieldname}</td>
<td>Fixed length year month day minute second (yyyymmddHHMMSS)</td>
<td>$d{time}, $d{departuretime}</td>
</tr>
<tr>
<td>Boolean</td>
<td>$b{fieldname}</td>
<td>The value is expressed as &quot;1&quot; if true and &quot;0&quot; if false, Other cases treated as NULL</td>
<td>$b{condition}, $b{condition}</td>
</tr>
</tbody>
</table>

### Table 4.3: Specification of retrieving values from previous row

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric(num)</td>
<td>#(fieldname)</td>
<td>${amount}, ${stockprice}</td>
</tr>
<tr>
<td>Character string(str)</td>
<td>#s{fieldname}</td>
<td>$s{gender}, $s{gender}</td>
</tr>
<tr>
<td>Date(date)</td>
<td>#d{fieldname}</td>
<td>$d{date}, $d{orderdate}</td>
</tr>
<tr>
<td>Time(time)</td>
<td>#t{fieldname}</td>
<td>$d{time}, $d{departuretime}</td>
</tr>
<tr>
<td>Boolean(bool)</td>
<td>#b{fieldname}</td>
<td>$b{condition}, $b{condition}</td>
</tr>
</tbody>
</table>

### Table 4.4: Specification to retrieve values from previous row

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric(num)</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>Character string(str)</td>
<td>{s}</td>
<td>{s}</td>
</tr>
<tr>
<td>Date(date)</td>
<td>{d}</td>
<td>{d}</td>
</tr>
<tr>
<td>Time(time)</td>
<td>{t}</td>
<td>{d}</td>
</tr>
<tr>
<td>Boolean(bool)</td>
<td>{b}</td>
<td>{b}</td>
</tr>
</tbody>
</table>
4.8 Arithmetic Operators

The + and – arithmetic operators can be used on numeric format strings as well as date and character format strings. The data format is shown in Table 4.5.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Format</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition (+)</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; + num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>Addition of numeric values</td>
<td>1.5+2.3 (3.8)</td>
</tr>
<tr>
<td></td>
<td>str&lt;sub&gt;1&lt;/sub&gt; + str&lt;sub&gt;2&lt;/sub&gt;</td>
<td>Join character strings</td>
<td>“150”+“円” (“150 円”)</td>
</tr>
<tr>
<td></td>
<td>date + num</td>
<td>num days after date</td>
<td>0d20121130+2 (0d20121202)</td>
</tr>
<tr>
<td></td>
<td>time + num</td>
<td>num seconds after time</td>
<td>0t095959+2 (0t100001)</td>
</tr>
<tr>
<td>Substraction (–)</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; – num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>Subtraction of numeric values</td>
<td>1.5–2.3 (–1.8)</td>
</tr>
<tr>
<td></td>
<td>str&lt;sub&gt;1&lt;/sub&gt; – str&lt;sub&gt;2&lt;/sub&gt;</td>
<td>Remove substring (by greedy match algorithm)</td>
<td>“aababa”–“a” (“bb”)</td>
</tr>
<tr>
<td></td>
<td>date – num</td>
<td>num days before date</td>
<td>0d20121202–2 (0d20121130)</td>
</tr>
<tr>
<td></td>
<td>time – num</td>
<td>num seconds before time</td>
<td>0t100001–2 (0t095959)</td>
</tr>
<tr>
<td></td>
<td>date&lt;sub&gt;1&lt;/sub&gt; – date&lt;sub&gt;2&lt;/sub&gt;</td>
<td>Date difference</td>
<td>0d20121202–0d20121130 (2)</td>
</tr>
<tr>
<td></td>
<td>time&lt;sub&gt;1&lt;/sub&gt; – time&lt;sub&gt;2&lt;/sub&gt;</td>
<td>Time difference</td>
<td>0t095959–0t100001 (–2)</td>
</tr>
<tr>
<td>Multiplication (*)</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; * num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>multiply</td>
<td>10*2 (20)</td>
</tr>
<tr>
<td>Division (÷)</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; / num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>divide</td>
<td>10/2 (5)</td>
</tr>
<tr>
<td>Remainder (%)</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; % num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>remainder</td>
<td>10%3 (1)</td>
</tr>
<tr>
<td>Power (^)</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; ^ num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>power</td>
<td>10^3 (1000)</td>
</tr>
</tbody>
</table>

The results of the examples are shown in parentheses (the content is shown using constant numbers).

4.9 Comparison Operators

The comparison operators can only be used on data of the same type. Table 4.6 shows the list of operators for numeric format data. Similarly, the operators shown in the table below can be applied to character format, date format, time format data.

<table>
<thead>
<tr>
<th>Details of comparison</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; == num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>1.5==1.5(0b1), ”abc”==”abcd” (0b0)</td>
</tr>
<tr>
<td>Not equal</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; != num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>1.5!=1.5(0b0), “abc”!=”abcd” (0b1)</td>
</tr>
<tr>
<td>Greater than</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; &gt; num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>10&gt;5(0b1), “abc”&gt;”abcd” (0b0)</td>
</tr>
<tr>
<td>Less than</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; &lt; num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>10&lt;5(0b0), “abc”&lt;”abcd” (0b1)</td>
</tr>
<tr>
<td>Above</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; &gt;= num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>10&gt;=10(0b1), “a”&gt;=”a” (0b1)</td>
</tr>
<tr>
<td>Below</td>
<td>num&lt;sub&gt;1&lt;/sub&gt; &lt;= num&lt;sub&gt;2&lt;/sub&gt;</td>
<td>8&lt;=9(0b1), “a”&lt;=”a” (0b1)</td>
</tr>
</tbody>
</table>

The results of the examples are shown in parentheses (the content is shown using constant numbers).

4.10 Logical Operator

The usage of the three logical operators (conjunction, disjunction, exclusive or) is shown in Table 4.7. In addition the results of the combination of boolean values (1:true, 0:false) are shown in Table 4.8, Table 4.9 and Table 4.10.

<table>
<thead>
<tr>
<th>Description</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conjunction</td>
<td>bool&lt;sub&gt;1&lt;/sub&gt; &amp; bool&lt;sub&gt;2&lt;/sub&gt;</td>
<td>”abc”==”abc” &amp;&amp; ”xyz”==”abc” (0b0)</td>
</tr>
<tr>
<td>Disjunction</td>
<td>bool&lt;sub&gt;1&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>Exclusive or</td>
<td>bool&lt;sub&gt;1&lt;/sub&gt; ^^ bool&lt;sub&gt;2&lt;/sub&gt;</td>
<td>”abc”==”abc” ^^ ”xyz”==”abc” (0b1)</td>
</tr>
</tbody>
</table>

The results of the examples are shown in parentheses (the content is shown using constant numbers).
4.11 Operator Precedence

The operators in Table 4.11 are listed according to precedence order.

The order of precedence starts from the top, operators on the same line with equal precedence are evaluated from left to right. When operators of equal precedence appear in the same expression, use parentheses to change to operator precedence and the expression to evaluated first.

4.12 Function

The following highlights the 9 types of functions in relation to numeric strings (4.12), trigonometric function (), character strings (4.14), regular expression (4.15), date / time (4.16), logical (4.17), row/column information (4.18), Null value (4.19), data type conversion (4.20).

4.13 Date and Time Format

There are two data types in mcal, namely date and time format. One is the date type and the other are date type. Time formatted data is represented with date formatted data as a set. The command uses date_time library of boost C++ library based on the Gregorian calendar, date type uses class boost::gregorian::date, and time type uses class boost::posix_time::ptime. For more details, refer to documentation in boost.org.

Date class is managed as a 32-bit integer internally, and supports dates ranging from January 1,1400 to December 31, 9999. Operations on date is based on the Gregorian calendar. NULL value will be returned on invalid date (for example, 2013/2/29 or 1399/12/31).

On the other hand, class ptime is managed as 64 bit. It is a time system with nano-second/micro-second resolution. The mcal command do not have an interface for time point manipulation. Class ptime is dependent on gregorian::date for the interface to the date portion of a time point, thereby enable time calculations across different dates. NULL value is returned on invalid time (e.g.18:62:11).

MCMD deals with CSV text, date/time must be assigned as character string in the data. The command then converts character string to date and time type for processing various operations. The final result converted back to character string in the output. The string format is expressed as 8-digit fixed-length string for date (e.g. "20130911"), 14-digit fixed-length string for time (e.g. "20130911110528") or the standard 6-digit fixed-length string (for example, "110528").

Figure 4.1 below shows the relationship of date type, time type and various functions.

<table>
<thead>
<tr>
<th>Table 4.8: Conjunction</th>
<th>Table 4.9: Disjunction</th>
<th>Table 4.10: Exclusive Or</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bool1</strong></td>
<td><strong>bool2</strong></td>
<td><strong>Result</strong></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

| Table 4.11: Precedence of operators |
|---|---|
| **Order** | **Operator** |
| 1 | *, /, %, ^ |
| 2 | +, - |
| 3 | >, <, >=, <= |
| 4 | ==, != |
| 5 | && |
| 6 | ||, ^^ |
Other than using fixed length character string as a standard for date/time, user can use Julian day (e.g. continuous count of days since Julian period such as January 1, 4713 BC) or UNIX time (e.g. number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970) as a signed integer to represent date and time. The command supports Julian day and UNIX time, as well as conversion functions of date and time data type.

The mcal command uses an internal date/time format which is based on the Gregorian calendar, thus the date range is limited from January 1,1400 to December 31,9999. Since UNIX time is a signed integer data type of 32 bits, the furthest time that can be represented this way is 03:14:07 UTC on Tuesday, 19 January 2038, date beyond this point will be interpreted incorrectly due to integer overflow. The drawback of using UNIX time

---

Table 4.12: Summary of numerical functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Function name</th>
<th>Function</th>
<th>Output type</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.96</td>
<td>sum(num1, num2, ⋯)</td>
<td>Sum</td>
<td>num</td>
</tr>
<tr>
<td>4.22</td>
<td>avg(num1, num2, ⋯)</td>
<td>Average</td>
<td>num</td>
</tr>
<tr>
<td>4.95</td>
<td>sqsum(num1, num2, ⋯)</td>
<td>Sum of squares</td>
<td>num</td>
</tr>
<tr>
<td>4.66</td>
<td>min(num1, num2, ⋯)</td>
<td>Minimum value</td>
<td>num</td>
</tr>
<tr>
<td>4.64</td>
<td>max(num1, num2, ⋯)</td>
<td>Maximum value</td>
<td>num</td>
</tr>
<tr>
<td>4.76</td>
<td>product(num1, num2, ⋯)</td>
<td>Product</td>
<td>num</td>
</tr>
<tr>
<td>4.39</td>
<td>factorial(num)</td>
<td>Factorial</td>
<td>num</td>
</tr>
<tr>
<td>4.45</td>
<td>gcd(num1, num2)</td>
<td>Greatest common divisor</td>
<td>num</td>
</tr>
<tr>
<td>4.54</td>
<td>lcm(num1, num2)</td>
<td>Least common multiple</td>
<td>num</td>
</tr>
<tr>
<td>4.94</td>
<td>sqrt(num)</td>
<td>Square root</td>
<td>num</td>
</tr>
<tr>
<td>4.14</td>
<td>abs(num)</td>
<td>Absolute value</td>
<td>num</td>
</tr>
<tr>
<td>4.91</td>
<td>sign(num)</td>
<td>Sign</td>
<td>num</td>
</tr>
<tr>
<td>4.50</td>
<td>int(num)</td>
<td>Integer part</td>
<td>num</td>
</tr>
<tr>
<td>4.44</td>
<td>frac(num)</td>
<td>Fraction part</td>
<td>num</td>
</tr>
<tr>
<td>4.89</td>
<td>round(num,nominal value)</td>
<td>Rounding up</td>
<td>num</td>
</tr>
<tr>
<td>4.42</td>
<td>floor(num,nominal value)</td>
<td>Rounding down</td>
<td>num</td>
</tr>
<tr>
<td>4.26</td>
<td>ceil(num,nominal value)</td>
<td>Ceiling</td>
<td>num</td>
</tr>
<tr>
<td>4.75</td>
<td>power(num,exponent)</td>
<td>Power</td>
<td>num</td>
</tr>
<tr>
<td>4.38</td>
<td>exp(num)</td>
<td>Exponential function</td>
<td>num</td>
</tr>
<tr>
<td>4.60</td>
<td>log(num,base)</td>
<td>logarithm</td>
<td>num</td>
</tr>
<tr>
<td>4.59</td>
<td>ln(num)</td>
<td>Natural logarithm</td>
<td>num</td>
</tr>
<tr>
<td>4.62</td>
<td>log2(num)</td>
<td>Binary logarithm</td>
<td>num</td>
</tr>
<tr>
<td>4.61</td>
<td>log10(num)</td>
<td>Common logarithm</td>
<td>num</td>
</tr>
<tr>
<td>4.34</td>
<td>dist(type,num1,num2, ⋯)</td>
<td>Distance</td>
<td>num</td>
</tr>
<tr>
<td>4.35</td>
<td>distgps(latitude1,longtitude1,latitude2,longtitude2)</td>
<td>GPS distance</td>
<td>num</td>
</tr>
<tr>
<td>4.47</td>
<td>heron(num1,num2, ⋯)</td>
<td>Heron’s formula</td>
<td>num</td>
</tr>
<tr>
<td>4.78</td>
<td>rand([random seed])</td>
<td>Uniform random number</td>
<td>num</td>
</tr>
<tr>
<td>4.79</td>
<td>randi(minimum value, maximum value[, random seed])</td>
<td>Uniform random number</td>
<td>num</td>
</tr>
<tr>
<td>4.72</td>
<td>nrand(minimum value, maximum value[, random seed])</td>
<td>Normal random number</td>
<td>num</td>
</tr>
<tr>
<td>4.74</td>
<td>pi()</td>
<td>Pi</td>
<td>num</td>
</tr>
<tr>
<td>4.37</td>
<td>e()</td>
<td>Napier’s constant</td>
<td>num</td>
</tr>
<tr>
<td>4.43</td>
<td>format()</td>
<td>Format output</td>
<td>str</td>
</tr>
</tbody>
</table>

Table 4.13: List of trigonometric functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Function Name</th>
<th>Function</th>
<th>Output range</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.15</td>
<td>acos(num)</td>
<td>Inverse cosine</td>
<td>0 ~ π</td>
</tr>
<tr>
<td>4.19</td>
<td>asin(num)</td>
<td>Inverse sine</td>
<td>−π ~ π</td>
</tr>
<tr>
<td>4.20</td>
<td>atan(num)</td>
<td>Inverse tangent</td>
<td>−π ~ π</td>
</tr>
<tr>
<td>4.21</td>
<td>atan2(num1,num2)</td>
<td>Angle of coordinates (num1, num2)</td>
<td>−π ~ π</td>
</tr>
<tr>
<td>4.27</td>
<td>cos(r)</td>
<td>Cosine</td>
<td>−1.0 ~ 1.0</td>
</tr>
<tr>
<td>4.92</td>
<td>sin(r)</td>
<td>Sine</td>
<td>−1.0 ~ 1.0</td>
</tr>
<tr>
<td>4.97</td>
<td>tan(r)</td>
<td>Tangent</td>
<td>−∞ ~ ∞</td>
</tr>
<tr>
<td>4.32</td>
<td>degree(r)</td>
<td>Degree</td>
<td>−π ~ π</td>
</tr>
<tr>
<td>4.77</td>
<td>radian(angle)</td>
<td>Enter angle as input, return radian as output</td>
<td>−π ~ π</td>
</tr>
<tr>
<td>4.28</td>
<td>cosh(r)</td>
<td>Hyperbolic cosine</td>
<td>0 ~ ∞</td>
</tr>
<tr>
<td>4.93</td>
<td>sinh(r)</td>
<td>Hyperbolic sine</td>
<td>−∞ ~ ∞</td>
</tr>
<tr>
<td>4.98</td>
<td>tanh(r)</td>
<td>Hyperbolic tangent</td>
<td>−1.0 ~ 1.0</td>
</tr>
</tbody>
</table>

Radian is represented by the variable r.
and Julian day is that one will not be able to tell the date and time by looking at the number.

Table 4.14: Character string related functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Function Name</th>
<th>Function</th>
<th>Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.25</td>
<td><code>cat(token, str_1, str_2, ...)</code></td>
<td>Merge character string</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.57</td>
<td><code>length(str)</code></td>
<td>Length of character string</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.40</td>
<td><code>fixlen(str, length, position, padding character)</code></td>
<td>Fixed length conversion</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.88</td>
<td><code>right(str, length)</code></td>
<td>Extract substring from the end</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.56</td>
<td><code>mid(str, starting position, length)</code></td>
<td>Extract substring from the beginning</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.103</td>
<td><code>toupper(str)</code></td>
<td>Convert characters from lowercase to uppercase</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.101</td>
<td><code>tolower(str)</code></td>
<td>Converts characters from uppercase to lowercase</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.24</td>
<td><code>capitalize(str)</code></td>
<td>Capitalize the first character</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.63</td>
<td><code>match(search string, str_1, str_2, ...)</code></td>
<td>Search for matched strings</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.46</td>
<td><code>haspace(str)</code></td>
<td>Search for white-space characters</td>
<td><code>bool</code></td>
</tr>
</tbody>
</table>

Table 4.15: Regular expression related functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Function Name</th>
<th>Function</th>
<th>Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.81</td>
<td><code>regexm(str, regular expression)</code></td>
<td>Match whole string</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.85</td>
<td><code>regexs(str, regular expression)</code></td>
<td>Match</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.84</td>
<td><code>regexrep(str, regular expression, replacement string)</code></td>
<td>Replace matching character string</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.80</td>
<td><code>regexlen(str, regular expression)</code></td>
<td>Match number of characters</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.83</td>
<td><code>regexpos(str, regular expression)</code></td>
<td>Start position of character</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.87</td>
<td><code>regextx(str, regular expression)</code></td>
<td>Match character string</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.82</td>
<td><code>regexpfx(str, regular expression)</code></td>
<td>Match prefix of character string</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.86</td>
<td><code>regexpfx(str, regular expression)</code></td>
<td>Match suffix of character string</td>
<td><code>str</code></td>
</tr>
</tbody>
</table>

Table 4.16: Date and Time Related Functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Function Name</th>
<th>Function</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.100</td>
<td><code>today()</code></td>
<td>Today’s date</td>
<td><code>date</code></td>
</tr>
<tr>
<td>4.70</td>
<td><code>now()</code></td>
<td>Current time</td>
<td><code>time</code></td>
</tr>
<tr>
<td>4.104</td>
<td><code>seconds(time)</code></td>
<td>Seconds elapsed</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.55</td>
<td><code>leapyear(dt)</code></td>
<td>Decide leap year</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.107</td>
<td><code>year(dt)</code></td>
<td>Gregorian calendar</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.68</td>
<td><code>month(dt)</code></td>
<td>Month</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.30</td>
<td><code>day(dt)</code></td>
<td>Day</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.106</td>
<td><code>week(dt)</code></td>
<td>Week number</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.36</td>
<td><code>dow(dt)</code></td>
<td>Day of week</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.99</td>
<td><code>time(time)</code></td>
<td>Hour minute second</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.31</td>
<td><code>date(time)</code></td>
<td>Year month day</td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.48</td>
<td><code>hour(time)</code></td>
<td>Hour</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.67</td>
<td><code>minute(time)</code></td>
<td>Minute</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.90</td>
<td><code>second(time)</code></td>
<td>Second</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.16</td>
<td><code>age(dt_1, dt_2)</code></td>
<td>Age</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.33</td>
<td><code>diff(dt_1, dt_2)</code></td>
<td>Period</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.105</td>
<td><code>uxt(dt)</code></td>
<td>Convert to UNIX time</td>
<td><code>num(UNIX time)</code></td>
</tr>
<tr>
<td>4.53</td>
<td><code>julian(dt)</code></td>
<td>Convert to Julian day</td>
<td><code>num(Julian day)</code></td>
</tr>
</tbody>
</table>

`dt` represents either `date` or `time`.

Table 4.17: Logical Functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Function Name</th>
<th>Function</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.17</td>
<td><code>and(book_1, book_2, ...)</code></td>
<td>Conjunction</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.73</td>
<td><code>or(book_1, book_2, ...)</code></td>
<td>Disjunction</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.69</td>
<td><code>not(book)</code></td>
<td>NOT</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.49</td>
<td><code>if(book, num_1, num_2)</code></td>
<td>Check logical condition</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.49</td>
<td><code>if(book, str_1, str_2)</code></td>
<td><code>str</code></td>
<td></td>
</tr>
<tr>
<td>4.49</td>
<td><code>if(book, date_1, date_2)</code></td>
<td><code>date</code></td>
<td></td>
</tr>
<tr>
<td>4.49</td>
<td><code>if(book, time_1, time_2)</code></td>
<td><code>time</code></td>
<td></td>
</tr>
</tbody>
</table>
Figure 4.1: The relationship among time type, date type and various functions using September 6, 2013 at 43 minutes, 27 seconds. The solid line box indicates actual data, the dotted line indicate the functions.

### Table 4.18: Row/column related functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Function Name</th>
<th>Function</th>
<th>Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.58</td>
<td><code>line()</code></td>
<td>Return the processing line number</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.102</td>
<td><code>top()</code></td>
<td>Top row</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.23</td>
<td><code>bottom()</code></td>
<td>Last row</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.41</td>
<td><code>fldsize()</code></td>
<td>Number of fields</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.18</td>
<td><code>argsize(str1, str2, ...)</code></td>
<td>Number of arguments</td>
<td><code>num</code></td>
</tr>
</tbody>
</table>

### Table 4.19: NULL value related functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Function Name</th>
<th>Function</th>
<th>Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.71</td>
<td><code>nulln()</code></td>
<td>NULL value</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.71</td>
<td><code>nulls()</code></td>
<td></td>
<td><code>str</code></td>
</tr>
<tr>
<td>4.71</td>
<td><code>nulld()</code></td>
<td></td>
<td><code>date</code></td>
</tr>
<tr>
<td>4.71</td>
<td><code>nullt()</code></td>
<td></td>
<td><code>time</code></td>
</tr>
<tr>
<td>4.71</td>
<td><code>nullb()</code></td>
<td></td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.52</td>
<td><code>isnull(num)</code></td>
<td>NULL value check</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.52</td>
<td><code>isnull(str)</code></td>
<td></td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.52</td>
<td><code>isnull(date)</code></td>
<td></td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.52</td>
<td><code>isnull(time)</code></td>
<td></td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.52</td>
<td><code>isnull(bool)</code></td>
<td></td>
<td><code>bool</code></td>
</tr>
<tr>
<td>4.29</td>
<td><code>countnull(num1, num2, ...)</code></td>
<td>Number of NULL values</td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.29</td>
<td><code>countnull(str1, str2, ...)</code></td>
<td></td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.29</td>
<td><code>countnull(date1, date2, ...)</code></td>
<td></td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.29</td>
<td><code>countnull(time1, time2, ...)</code></td>
<td></td>
<td><code>num</code></td>
</tr>
<tr>
<td>4.29</td>
<td><code>countnull(bool1, bool2, ...)</code></td>
<td></td>
<td><code>num</code></td>
</tr>
</tbody>
</table>

### Table 4.20: Type conversion related functions

<table>
<thead>
<tr>
<th>Section</th>
<th>num</th>
<th>str</th>
<th>date</th>
<th>time</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.108</td>
<td>n2s(num)</td>
<td>s2n(str)</td>
<td>d2s(date)</td>
<td>t2s(time)</td>
<td>b2n(bool)</td>
</tr>
<tr>
<td>num</td>
<td>n2b(num)</td>
<td>s2b(str)</td>
<td>s2b(str)</td>
<td>d2b(date)</td>
<td>b2b(bool)</td>
</tr>
<tr>
<td>str</td>
<td>d2s(date)</td>
<td>t2s(time)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>t2d(time)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>b2d(bool)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>b2s(bool)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each cell corresponds to the conversion function from the labels in the top row to labels in the left column. Empty cells means that conversion function is not available.
4.14 abs - Absolute value

Format: abs(num)
Compute absolute value of num.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,1.0
2,-2.5
3,
4,0
$ mcal c='abs(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=abs(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,1.0,1
2,2.5
3,
4,0,0
```
4.15 acos - Inverse Cosine (arccosine)

Format: acos(num)

Compute arccosine (inverse cosine). The function evaluates the principal value ranges of $-1.0 \sim 1.0$, and returns values within the parameter of $0.0 \sim \pi$.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,-1.0
2,0
3,1.0
4,
5,2
$ mcal c='acos(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END#
kgcal a=rsl c=acos(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,-1.0,3.141592654
2,0,1.570796327
3,1.0,0
4,,
5,2,
```
4.16 age - Age

Format 1: age(Birth date YYYYMMDD, date)
Format 2: age(Birth date YYYYMMDD, time)

Computes age with date (format 1) and time (format 2). Conversion of date or time values to birth date (in year, month, day format).

Example

Example 1: Basic Example

Compute the age from date of birth to a fixed point on calendar on September 1, 2013.

```
$ more dat1.csv
id,dob
1,19641010
2,20000101
3,,
4,19770812
$ mcal c='age($d{dob},0d20130901)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=age($d{dob},0d20130901) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,dob,rsl
1,19641010,48
2,20000101,13
3,,
4,19770812,36
```
4.17 and - Logical Conjunction

Format: \texttt{and(\texttt{bool}_1,\texttt{bool}_2,\cdots)}

Calculate the logical conjunction of boolean value \texttt{bool}_i. Please refer to Table 4.8 on boolean value table with NULL values.

Example

Example 1: Basic Example

```plaintext
$ more dat1.csv
id,b1,b2,b3
1,1,0,1
2,1,1,1
3,1,,1
4,1,1,1
$ mcal c='and($b{b1},$b{b2},$b{b3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=and($b{b1},$b{b2},$b{b3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,b1,b2,b3,rsl
1,1,0,1,0
2,1,1,1,1
3,1,,1,
4,1,1,1,1
```

Example 2: Example of using wildcard

Specify columns names that start with \texttt{b} as in (\texttt{b1},\texttt{b2},\texttt{b3}) using a wildcard in column name such as "\texttt{b*}".

```plaintext
$ mcal c='and($b{b*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=and($b{b*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,b1,b2,b3,rsl
1,1,0,1,0
2,1,1,1,1
3,1,1,
4,1,1,1,1
```

4.18 argument - Number of Arguments

Format 1: `argsize(str_1, str_2, ...)`

Return the number of character strings specified by `str_i`. The column names can be specified using wildcard. It is possible to count the number of items using wild card. An important point to note is that this function is only compatible with character string format.

Example 1: Basic Example

Count the number of column names that start with "v".

```bash
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,,,
$ mcal c='argsize(${v*})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=argsize(${v*}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,3
2,-5,2,1,3
3,1,,3,3
4,,,,3
```
4.19 asin - Inverse Sine

Format: asin(num)

Compute arcsine (inverse sine). The function evaluates the principal value ranges from \(-1.0 \sim 1.0\), and returns values within the parameter of \(-\pi/2 \sim \pi/2\).

Example

Example 1: Basic Example

```bash
$ more dat1.csv
id,val
1,-1.0
2,0
3,1.0
4,
5,2
$ mcal c='asin($val))' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=asin($val)) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,-1.0,-1.570796327
2,0,0
3,1.0,1.570796327
4,2,
5,2,
```
4.20 atan - Inverse Tangent

Format: atan(num)

Compute arctangent (inverse tangent). The function evaluates the principal value ranges from $-\infty \sim \infty$, and returns values within the parameter of $-\pi/2 \sim \pi/2$.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,-1.0
2,0
3,1.0
4,
5,1.0e+10
$ mcal c='atan(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=atan(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,-1.0,-0.7853981634
2,0,0
3,1.0,0.7853981634
4,
5,1.0e+10,1.570796327
```
4.21 atan2 - Angle of coordinates

Format: atan2(num₁, num₂)

Return the angle in radians formed by the axis and line segment of x, y coordinates (num₁, num₂) and origin.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,x,y
1,5,10
2,10,20
3,-1,0
4,0,0
5,,
$ mcal c='atan2(x,y)' a=rsl i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,x,y,a
1,5,10,1.107148718
2,10,20,1.107148718
3,-1,0,3.141592654
4,0,0,0
5,,
```
4.22 avg - Average

Format: \texttt{avg(num_1, num_2, \cdots)}

Compute the average of numbers given in \textit{num}_i. The function ignore NULL values, and return NULL result if all values in input is NULL.

Examples

Example 1: Basic Example

\begin{verbatim}
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,,
$ mcal c='avg(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=avg(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,2
2,-5,2,1,-0.6666666667
3,1,,3,2
4,,,
\end{verbatim}

Example 2: Example of using wildcard

Specify columns names that start with \textit{v} (v1,v2,v3) using a wildcard in column name such as \textit{v*}.

\begin{verbatim}
$ mcal c='avg(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=avg(${v*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,2
2,-5,2,1,-0.6666666667
3,1,3,2
4,,
\end{verbatim}
4.23  bottom - Last row

Format: bottom()

Return true if record is positioned in the last row, otherwise return false.

Example

Example 1: Basic Example

```
$ more dat1.csv
val
1
2
3
4
$ mcalf c='bottom()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=bottom() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
val,rsl
1,0
2,0
3,0
4,1
```
4.24 **capitalize - Capitalize first character**

Format: capitalize(str)

Change the first character of the string to uppercase. This function does not affect non-alphabet strings.

**Example**

**Example 1: Basic Example**

Convert the first character of str field to uppercase.

```sh
$ more dat1.csv
id,str
1,abc
2,aBd
3, 
4,#abc

$ mcal c='capitalize($s{str})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=capitalize($s{str}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abc,Abc
2,aBd,ABd
3, ,
4,#abc,#abc
```
### 4.25 cat - Concatenate Character String

**Format:** cat(token, str1, str2, ...)

Concatenate the specified str in order using token as the delimiter to create a character string. If an empty character "" is specified, the list of strings are simply concatenated.

#### Examples

**Example 1: Basic Example**

Concatenate 3 columns str1, str2, str3 with the inclusion of "#" as delimiter token in between characters.

```
$ more dat1.csv
id,str1,str2,str3
1,abc,def,ghi
2,A,,CDE
3,,
4,,XY
$ mcal c='cat("#",$s{str1},$s{str2},$s{str3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=cat("#",$s{str1},$s{str2},$s{str3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str1,str2,str3,rsl
1,abc,def,ghi,abc#def#ghi
2,A,,CDE,A##CDE
3,,,##
4,,,XY,##XY
```

**Example 2: Empty token**

```
$ mcal c='cat("",$s{str1},$s{str2},$s{str3})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=cat("",$s{str1},$s{str2},$s{str3}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,str1,str2,str3,rsl
1,abc,def,ghi,abcdefghi
2,A,,CDE,ACDE
3,,,##
4,,,XY,XY
```

**Example 3: Example using wildcard**

Use wildcard to specify columns names that start with str (str1, str2, str3) such as str*.

```
$ mcal c='cat("",$s{str*})' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=cat("",$s{str*}) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,str1,str2,str3,rsl
1,abc,def,ghi,abcdefghi
2,A,,CDE,ACDE
3,,,##
4,,,XY,XY
```
4.26 ceil - Ceiling

Format: ceil(num,base)

Round up num based on multiples of the base number, and returns the smallest integer that is greater than or equal to num.

For example, given ceil(3.42,0.5), the numerical value will be rounded up by scales of 0.5 that is greater than 3.42. Thus, the rounded number at base 0.5 becomes 3.5. When base is not specified as an argument, the default base is set at 1. This is equivalent to rounding up to the nearest integer after the decimal.

Examples

Example 1: Basic Example

Truncate all digits after decimal point.

```bash
$ more dat1.csv
id,val
1,3.28
2,3.82
3,4,-0.6
$ mcal c='floor(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=floor(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.28,3
2,3.82,3
3,4,-0.6,-1
```

Example 2: Basic Example

Truncate the second digit after decimal point.

```bash
$ mcal c='floor(${val},0.1)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=floor(${val},0.1) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,3.28,3.2
2,3.82,3.8
3,,
4,-0.6,-0.6
```

Example 3: Round to base 0.5

Rounding to the nearest 0.5.

```bash
$ mcal c='floor(${val},0.5)' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=floor(${val},0.5) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,val,rsl
1,3.28,3
2,3.82,3.5
3,,
4,-0.6,-1
```

Example 4: Round to base 10

Rounding to the nearest 10th digit.
$ more dat2.csv
id,val
1,1341.28
2,188
3,1.235E+3
4,-1.235E+3
$ mcal c='floor(${val},10)' a=rsl i=dat2.csv o=rsl4.csv
#END# kgcal a=rsl c=floor(${val},10) i=dat2.csv o=rsl4.csv
$ more rsl4.csv
id,val,rsl
1,1341.28,1340
2,188,180
3,1.235E+3,1230
4,-1.235E+3,-1240
4.27 cos - Cosine

Format: \( \cos(r) \)

Compute radians of \( r \) using cosine function.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.141592
2,1.047197
3,
4,6.283185
$ mcal c='\cos(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=\cos(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,-1
2,1.047197,0.5000004774
3,,
4,6.283185,1
```
4.28  cosh - Hyperbolic Cosine

Format: cosh(r)
Compute hyperbolic cosine.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.141592
2,-1.047197
3,
4,6.283185
$ mcal c='cosh(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=cosh(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,11.59194573
2,-1.047197,1.600286169
3,
4,6.283185,267.7466792
```
4.29 countnull - Sum total

Format 1: countnull(num1, num2, ⋯)
Format 2: countnull(str1, str2, ⋯)
Format 3: countnull(date1, date2, ⋯)
Format 4: countnull(time1, time2, ⋯)
Format 5: countnull(bool1, bool2, ⋯)

Return the number of NULL values in numi (same for other types).

Examples

Example 1: Basic Example

```bash
$ more dat1.csv
a,b,c,d
1,,3,4
1,,,

$ mcal c='countnull(${a},${b},${c},${d})' a=rsl i=dat1.csv o=rsl1.csv
$ more rsl1.csv
a,b,c,d,rsl
1,,3,4,1
1,,,,3
,,,,4
```

Example 2: Specify other types of data format in field with wildcard character

```bash
$ mcal c='countnull($s{*})' a=rsl i=dat1.csv o=rsl2.csv
$ more rsl2.csv
a,b,c,d,rsl
1,,3,4,1
1,,,3
,,,,,4
```
4.30  day - Day

Format 1: day(date)
Format 2: day(time)
Format 3: days(date)
Format 4: days(time)

Extract day from time or date. The function shown in format 1 and 2 returns the numeric value, the function in format 3 and 4 returns 2-digit fixed-length string.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,date
1,20000101
2,20121021
3,19770812
$ mcal c='day($d{date})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=day($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,1
2,20121021,21
3,19770812,12
```

Example 2: Time format

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,19770812122212
$ mcal c='day($t{time})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=day($t{time}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,1
2,20121021111213,21
3,19770812122212,12
```
4.31 date - Year Month Day

Format: date(time)

Extract 8-digit fixed length string of Year Month Day (YYYYMMDD) from time.

Example

Example 1: Basic Example

```sh
$ more dat1.csv
id,time
1,20000101000000
2,20121021111113
3,
4,19770812122212
$ mcal c='date($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=date($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,20000101000000,20000101
2,20121021111113,20121021
3,
4,19770812122212,19770812
```
4.32 degree - Degree

Format: degree(r)

Compute the degree corresponding to radian r.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.141592
2,1.047197
3,
4,6.283185
$ mcalkcal c='degree(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kcalkal a=rsl c=degree(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,179.9999626
2,1.047197,59.99996842
3,
4,6.283185,359.9999824
```
4.33  **diff - Calculate Interval**

Format 1: `diffyear(date_1, date_2)`
Format 2: `diffyear(time_1, time_2)`
Format 3: `diffmonth(date_1, date_2)`
Format 4: `diffmonth(time_1, time_2)`
Format 5: `diffday(date_1, date_2)`
Format 6: `diffday(time_1, time_2)`
Format 7: `diffhour(date_1, date_2)`
Format 8: `diffhour(time_1, time_2)`
Format 9: `diffminute(date_1, date_2)`
Format 10: `diffminute(time_1, time_2)`
Format 11: `diffsecond(date_1, date_2)`
Format 12: `diffsecond(time_1, time_2)`

Compute the interval between two dates or time such as `date_1(time_1)` and `date_2(time_2)`, where the interval is measured in terms of months, days, hours, minutes, seconds. Fractions are rounded up.

**Examples**

**Example 1: Compute the interval in terms of month**

Compute the number of months during the period between the value in the `date` column and September 1, 2013.

```bash
$ more dat1.csv
id,date
1,19641010
2,20000101
3,4,19770812
$ mcal c='diffmonth($d{date},0d20130901)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=diffmonth($d{date},0d20130901) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,19641010,-587
2,20000101,-164
3,4,19770812,-433
```

**Example 2: Compute the interval in terms of minutes**

Compute the minutes between the value in the `time` column and January 1, 2012 00 hours 00 minutes 00 seconds.

```bash
$ more dat2.csv
id,time
1,20120101000000
2,20120101011112
3,4,20111231235000
$ mcal c='diffmonth($t{time},0t20120101000000)’ a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=diffmonth($t{time},0t20120101000000) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20120101000000,0
2,20120101011112,0
3,4,20111231235000,-1
```
4.34 dist - Distance

Format: dist(type, num1, num2, ..., nk, numk+1, numk+2, ..., num2k)

Compute the distance between two dimension vectors (num1, num2, ..., nk), (numk+1, numk+2, ..., num2k). Refer to msim for detailed definitions.

- euclid: Euclidean distance
- cityblock: City Block distance
- hamming: Hamming distance

Hamming distance must be specified in character string format (refer to the example below).

Examples

Example 1: Euclidean distance

```
$ more dat1.csv
id,x1,y1,x2,y2
1,0,0,1,1
2,0,1,2,0
3,...
$ mcal c='dist("euclid",${x1},${y1},${x2},${y2})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=dist("euclid",${x1},${y1},${x2},${y2}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,x1,y1,x2,y2,rsl
1,0,0,1,1,1.414213562
2,0,1,2,0,2.236067977
3,...
```

Example 2: City Block distance

```
$ mcal c='dist("cityblock",${x1},${y1},${x2},${y2})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=dist("cityblock",${x1},${y1},${x2},${y2}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,x1,y1,x2,y2,rsl
1,0,0,1,1,2
2,0,1,2,0,3
3,...
```

Example 3: Hamming distance

Hamming distance must be specified in character string format for the calculation of Hamming distance.

```
$ more dat2.csv
id,x1,y1,x2,y2
1,a,b,a,c
2,0,1,0,1
3,...
$ mcal c='dist("hamming",${x1},${y1},${x2},${y2})' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=dist("hamming",${x1},${y1},${x2},${y2}) i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,x1,y1,x2,y2,rsl
1,a,b,a,c,1
2,0,1,0,1,2
3,...
```
4.35 distgps - GPS Distance

Format: distgps(latitude1,longitude1,latitude2,longitude2, [orientation])

Find out the straight-line / direct distance (km unit) between two points based on the latitude and longitude coordinates. The latitude and longitude can be expressed in signed decimal degrees without compass direction, where positive indicates north/east, negative indicates west/south, on the basis of a spherical earth. The latitude and longitude is represent in heading of 60 degrees, and the format is expressed as degree, minute, second (d; m; s). The values must be converted to base of 10 for use with this function. Base of 10 coordinates can be calculated by \( d + m/60 + s/600 \).

For example, the distance between Osaka (north latitude 34.702398, east longitude 135.495188) and Tokyo (north latitude 35.681391, east longitude 139.766103) is calculated as follows.

\[
\text{distgps}(34.702398, 135.495188, 35.681391, 139.766103)
\]

In addition, the distance from Everest (north latitude 32.655556, east longitude 79.015833) to Aconcagua (southern latitude 27.987778, west longitude 86.944444) is specified as follows.

\[
\text{distgps}(32.655556, 79.015833, -27.987778, -86.944444)
\]

Example

Example 1: Basic Example

```
$ more dat1.csv
point1,point2,lat1,lon1,lat2,lon2
osaka,tenma,34.702398,135.495188,34.704923,135.512233
osaka,tokyo,34.702398,135.495188,35.681391,139.766103
osaka,kobe,34.702398,135.495188,34.679453,135.178221
osaka,Fuji,34.702398,135.495188,35.360556,138.727500
Evelest,Aconcagua,32.655556,79.015833,-27.987778,-86.944444
Denali,Kilimanjaro,63.069444,-151.007222,-3.075833,37.353333
$ mcal c='distgps(${lat1},${lon1},${lat2},${lon2})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=distgps(${lat1},${lon1},${lat2},${lon2}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
point1,point2,lat1,lon1,lat2,lon2,rsl
osaka,tenma,34.702398,135.495188,34.704923,135.512233,1.585046048
osaka,tokyo,34.702398,135.495188,35.681391,139.766103,405.774306
osaka,kobe,34.702398,135.495188,34.679453,135.178221,29.12042213
osaka,Fuji,34.702398,135.495188,35.360556,138.727500,304.7527532
Evelest,Aconcagua,32.655556,79.015833,-27.987778,-86.944444,16956.12242
Denali,Kilimanjaro,63.069444,-151.007222,-3.075833,37.353333,11362.37758
```
4.36 dow - Day of week

Format 1: dow(dt) Number of day of week number (1-7)
Format 2: dowj(dt) Day of week in Japanese
Format 3: dowe(dt) Day of week in English
Format 4: dowes(dt) Day of week abbreviation in English

Return the day of week from date and time. The notions of day of week is indicated in format 1 to 4 above. Weekday number is follows the ISO8601 date and time standard, number 1 to 7 corresponds to Monday to Sunday.

Examples

Example 1: Basic Example

```bash
$ more dat1.csv
id,date
1,20000101
2,20121021
3,4,19770812
$ mcal c='dow($d{date})' a=rsl i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,6
2,20121021,7
3,4,19770812,5
```

Example 2: Weekday in Japanese

```bash
$ mcal c='dowj($d{date})' a=rsl i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,date,rsl
1,20000101,月
2,20121021,日
3,4,19770812,金
```

Example 3: Weekday in English

```bash
$ mcal c='dowe($d{date})' a=rsl i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,date,rsl
1,20000101,Saturday
2,20121021,Sunday
3,4,19770812,Friday
```

Example 4: Weekday Abbreviated in English

```bash
$ mcal c='dowes($d{date})' a=rsl i=dat1.csv o=rsl4.csv
$ more rsl4.csv
id,date,rsl
1,20000101,Sat
2,20121021,Sun
```
Example 5: Time format

```
$ more dat2.csv
id,time
1,20000101000000
2,201210211111213
3,4,19770812122212
$ mcal c='dow($t{time})' a=rsl i=dat2.csv o=rsl5.csv
#END# kgcal a=rsl c=dow($t{time}) i=dat2.csv o=rsl5.csv
$ more rsl5.csv
id,time,rsl
1,20000101000000,6
2,201210211111213,7
3
4,19770812122212,5
```
4.37  e - Napier is Constant

Format: e()

Return Napier’s Constant (e).

Example

Example 1: Basic Example

```
$ more dat1.csv
id
1
2
$ mcald c='e()' a=rsl i=dat1.csv o=rsl1.csv
#END# kcal a=rsl c=e() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,2.718281828
2,2.718281828
```
4.38 \textit{exp - Exponential Function}

Format: \textit{exp(num)}

A type of power function using \textit{e} (Napier’s constant) as base to compute \textit{num}.

Example

Example 1: Basic Example

```bash
$ more dat1.csv
id,exponent
1,1
2,-1
3,0
4,0.5
$ mcal c='exp($\{\text{exponent$\})$' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcsl a=rsl c=exp($\{\text{exponent$\})$ i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,exponent,rsl
1,1,2.718281828
2,-1,0.3678794412
3,,
4,0,1
5,0.5,1.648721271
```
4.39 factorial - Factorial

Format: factorial(num)

Returns the factorial of num. NULL value is returned if the result exceeds the maximum value of real number.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1.1
2.5
3
4,10000
$ mcal c='factorial(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=factorial(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,1,1
2.5,120
3,,
4,10000,
```

Example 2: Example of using constants

Calculate the factorial of 5. When constants is used as an argument, all rows will return the same result.

```
$ mcal c='factorial(5)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=factorial(5) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,1,120
2.5,120
3,,120
4,10000,120
```
4.40 fixlen - Convert of Fixed Length String

Format 1: length(str,length,position,padding character)

Format 2: lengthw(str,length,position,padding character)

Convert str to fixed length string. Specify padding characters for left justified or right justified str if it is less than specified length. Specify the justification using "L" or "R" in the position parameter. Define the character to be embedded in the padding character parameter. Note that if the length of str exceeds the defined length, the ending characters for right justified character strings or beginning characters for left justified character strings will be cut off.

Use fixlenw function for fixed-length conversion on multi-byte characters.

Examples

Example 1: Basic Example

Convert values in the str column to 5 character fixed-length string. Right justified ("R") the strings and fill the empty positions with "#" for strings with less than 5 characters.

```
$ more dat1.csv
id, str
1, abc
2, 123
3, 41234567
$ mcal c='fixlen($s{str},5,"R","#")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=fixlen($s{str},5,"R","#") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id, str, rsl
1, abc, ##abc
2, 123, ##123
3, ####
4, 1234567, 34567
```

Example 2: Left justified

Fill empty positions for left justified ("L") text with "#".

```
$ mcal c='fixlen($s{str},5,"L","#")' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=fixlen($s{str},5,"L","#") i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id, str, rsl
1, abc, abc##
2, 123, 123##
3, ####
4, 1234567, 12345
```
4.41  **fldsize - Number of Fields**

Format: `fldsize()`

Return the number of fields in the input file. MCMD assumes all records in the input data has same number of fields, thus the resulting value from the function applies to all records.

**Examples**

**Example 1: Basic Example**

```bash
$ more dat1.csv
a,b,c,d
ga,b,c,d
b,c,d,e
$ more rsl1.csv
1,2,3,4,4
2,3,4,5,4
3,...,4
4,x,y,z,4
```

```bash
$ mcal c='fldsize()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=fldsize() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
a,b,c,d,rsl
1,2,3,4,4
2,3,4,5,4
3,...,4
4,x,y,z,4
```
4.42 floor - Rounding Down

Format: floor(num,base)

Round num down to the nearest integer. This function round down to the maximum integer that is not greater than num. For example, in the argument floor(3.82,0.5), the decimal point of 3.82 is above the scale point of 0.5, thus, rounding down to the nearest 0.5 base returns 3.5. The default value of base is 1 if the argument is not defined. This is equivalent to rounding to an integer value by truncating all decimal digits.

Examples

Example 1: Basic Example

Truncate all digits after decimal point.

```bash
$ more dat1.csv
id,val
1,3.28
2,3.82
3,-0.6
$mcal c='floor(${val})' a=rsl i=dat1.csv o=rsl1.csv
END# kgcal a=rsl c=floor(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.28,3
2,3.82,3
3,-0.6,-1
```

Example 2: Basic Example

Truncate the second digit after decimal point.

```bash
$mcal c='floor(${val},0.1)' a=rsl i=dat1.csv o=rsl2.csv
END# kgcal a=rsl c=floor(${val},0.1) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,3.28,3.2
2,3.82,3.8
3,-0.6,-0.6
```

Example 3: Round to base 0.5

Rounding to the nearest 0.5.

```bash
$mcal c='floor(${val},0.5)' a=rsl i=dat1.csv o=rsl3.csv
END# kgcal a=rsl c=floor(${val},0.5) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,val,rsl
1,3.28,3
2,3.82,3.5
3,-0.6,-1
```

Example 4: Round to base 10

Rounding to the nearest 10th digit.

```bash
$ more dat2.csv
id,val
1,1341.28
2,188
3,1.235E+3
```
4.42. FLOOR - ROUNDING DOWN

```bash
4.1235E+3
$ mcal c='floor(${val},10)' a=rsl i=dat2.csv o=rsl4.csv
#END# kgcal a=rsl c=floor(${val},10) i=dat2.csv o=rsl4.csv
$ more rsl4.csv
id, val, rsl
1, 1.341 28, 1340
2, 188, 180
3, 1.235E+3, 1230
4, -1.235E+3, -1240
```
246

4.43

CHAPTER 4. MCAL

format - Formatted Output

Format: format(num, output format)
mcal internally manages floating point numbers. This function changes the output format of the number
by specifying the output format available in the C language printf function. The 3 available formats are as
follows.
• %f: Decimal format
• %e,%E: Exponent format (Specify %E to use uppercase E as exponent notation)
• %g,%G: Automatically select output format of f or e (Specify %G for uppercase notation)
For example, the decimal format is expressed as 0.00726 can be expressed as 7.260000e-03 in exponent format,
similarly, the exponent format of 1265 is represented as 1.265000e+03.
By specifying the % or plus sign after the number, the function formats the decimal digits and characters.
The number format can be specified in the format of whole number.decimal . For example, the number
123.456789 with the format specifier of %5.2f becomes 123.46, and the format specifier of %8.3f print number
as 123.458 with a floating point of 8 characters and 3 characters after decimal.
Add a plus sign before the number to display the plus sign before the number. For example, 123.456789 with
the format specifier of %+5.2f format the number as +123.46.
Besides the format highlighted above, it is possible to include any character string in the format. For example,
the number 250 can be formatted as Total 250 Yen using the format specifier of Total

Examples
Example 1: Basic Example
Format val as real number with 3 decimal places.
$ more dat1.csv
id,val
1,0.00726
2,123.456789
3,
4,-0.335
$ mcal c=’format(${val},"%8.3f")’ a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=format(${val},"%8.3f") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,0.00726,
0.007
2,123.456789, 123.457
3,,
4,-0.335, -0.335

Example 2: Display the exponent
Format val in exponential notation.
$ mcal c=’format(${val},"%e")’ a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=format(${val},"%e") i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,0.00726,7.260000e-03
2,123.456789,1.234568e+02
3,,
4,-0.335,-3.350000e-01


4.44  fract - Fractional Part

Format: fract(num)

Returns the fractional part of num. Fractions of values in scientific notation will be output as is.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.14
2,3
3,
4,-12.56789
5,1.2345e+2
6,1.2345e-10
$ mcal c='fract(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kcal a=rsl c=fract(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.14,0.14
2,3,0
3,
4,-12.56789,-0.56789
5,1.2345e+2,0.45
6,1.2345e-10,1.2345e-10
```
4.45 gcd - Greatest Common Divisor

Format: gcd(num1, num2)

Find out the greatest common divisor of num1 and num2. Real number will be rounded down to whole number.

Example

Example 1: Basic Example

```bash
$ more dat1.csv
id,val1,val2
1,12,36
2,6,6
3,,
4,12.1,36.2
$ mcal c='gcd(${val1},${val2})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=gcd(${val1},${val2}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val1,val2,rsl
1,12,36,12
2,6,6,1
3,,
4,12.1,36.2,12
```

Example 2: Example of constants

Find out the greatest common divisor of val1 column and 36.

```bash
$ mcal c='gcd(${val1},36)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=gcd(${val1},36) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val1,val2,rsl
1,12,36,12
2,6,6,1
3,,
4,12.1,36.2,12
```
4.46  haspace - Search Whitespace Characters

Format 1: haspace(str, length)

Format 2: haspacew(str, length)

This function returns true if there is whitespace characters in character string str and false if not. Whitespace characters is represented as ASCII code 0x20 to 0x09-0x0d. The respective ASCII code are represented as: single byte space (0x20), horizontal tab (0x09), new line (0x0a), vertical tabulation (0x0b), new page (0x0c), carriage return (0x0d). Use haspacew to search for multibyte white space character.

Examples

Example 1: Basic Example

Returns true if str column contains white space characters. The first row where id=1 contains single-byte space character, in addition, the second row where id=2 contains tab character, thus, these two rows return true. However, records where the function returns false are id=4 which contains carriage return, and id=3 which contains double-byte space character.

```bash
$ more dat1.csv
id,str
1,a b
2,ab c
3,ab c
4,\
5,"aa bb"

$ mcal c='hasspace($s{str})' a=rsl i=dat1.csv o=rsl1.csv
END# kgcal a=rsl c=hasspace($s{str}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,a b,1
2,ab c,1
3,ab c,0
4,,0
5,"aa bb",1
```

Example 2: Multibyte character

Use haspacew function to detect double character space.

```bash
$ mcal c='hasspacew($s{str})' a=rsl i=dat1.csv o=rsl2.csv
END# kgcal a=rsl c=hasspacew($s{str}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,a b,1
2,ab c,1
3,ab c,0
4,,0
5,"aa bb",1
```
4.47  heron - Area of Triangle

Format: sim(type,num1,num2,⋯,nk,numk+1,numk+2,⋯,num2k,num2k+1,num2k+2,⋯,num3k)

Computes the area of a triangle given the 3 points of coordinates at k-dimensional space \((num_1, num_2, ⋯, num_k), (num_{k+1}, num_{k+2}, ⋯, num_{2k})\) at k-dimensional space.

Example

Example 1: Basic Example

```bash
$ more dat1.csv
id,x1,y1,x2,y2,x3,y3
1,0,0,1,0,0,1
2,0,0,2,2,0
4,\ldots,\ldots
3,0,0,1,1,2,2
$ mcal c='heron(${x1},${y1},${x2},${y2},${x3},${y3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=heron(${x1},${y1},${x2},${y2},${x3},${y3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,x1,y1,x2,y2,x3,y3,rsl
1,0,0,1,0,0,1,0.5
2,0,0,2,2,0,2
4,\ldots,\ldots
3,0,0,1,1,2,2,0
```
4.48. **HOUR - HOUR**

**4.48 hour - Hour**

Format 1: `hour(time)` Numerical value

Format 2: `hours(time)` 2 digit fixed length string

Extract hour from `time`. The function can be used in different purposes as shown in format 1,2.

**Examples**

**Example 1: Basic Example**

```bash
$ more dat1.csv
id,time
1,20000101000000
2,20121021111213
3,4,19770812122212
$ mcal c='hour($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=hour($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,20000101000000,0
2,20121021111213,11
3,4,19770812122212,12
```

**Example 2: Output character string**

```bash
$ mcal c='hours($t{time})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=hours($t{time}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,00
2,20121021111213,11
3,4,19770812122212,12
```
4.49  if - Conditional Statements

Format: if(bool, num1, num2), if(bool, str1, str2), if(bool, date1, date2), if(bool, time1, time2), if(bool, bool1, bool2)

If the first parameter is true, return the second parameter, otherwise, return the third parameter. If the first parameter includes NULL value, return NULL value. Note that second and third argument should use the same parameter type.

Examples

Example 1: Basic Example

If the value in time column is less than 120000, return "AM", otherwise, return "PM".

```bash
$ more dat1.csv
id,time
1,101215
2,210110
3,,
4,120000
$ mcal c='if(${time}<=120000,"AM","PM")' a=ampm i=dat1.csv o=rsl1.csv
#END# kgcal a=ampm c=if(${time}<=120000,"AM","PM") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,ampm
1,101215,AM
2,210110,PM
3,,
4,120000,PM
```

Example 2: Different parameter types

The function returns an error if the character format is different in the second and third parameter.

```bash
$ mcal c='if(${time}<=120000,"am",1)' a=ampm i=dat1.csv o=rsl2.csv
#ERROR# unknown function or operator: if_BSN (kgcal)
$ more rsl2.csv
```

Example 3: Return boolean value on conditional statements

Read the value in the column using $b{fieldname}, if the value is 1 return true, if it is 0 return false, other values will be treated as NULL.

```bash
$ more dat2.csv
id,val
1,1
2,0
3,,
4,-2
$ mcal c='if($b{val},"true","false")' a=bool i=dat2.csv o=rsl3.csv
#END# kgcal a=bool c=if($b{val},"true","false") i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,val,bool
1,1,true
2,0,false
3,,
4,-2,
```

Example 4: Time format comparison

```bash
$ mcal c='if(${t{time}}<=0t120000,"am","pm")' a=ampm i=dat1.csv o=rsl4.csv
#END# kgcal a=ampm c=if(${t{time}}<=0t120000,"am","pm") i=dat1.csv o=rsl4.csv
$ more rsl4.csv
id,time,ampm
1,101215,am
2,210110,pm
```
Example 5: Nested if function

```bash
$ more dat3.csv
id,val
1,10
2,0
3,-5
4,0

$ mcal c='if(${val}>0,"plus",if(${val}<0,"minus","zero"))' a=sign i=dat3.csv o=rsl5.csv
#END# kgcal a=sign c=if(${val}>0,"plus",if(${val}<0,"minus","zero")) i=dat3.csv o=rsl5.csv

$ more rsl5.csv
id,val,sign
1,10,plus
2,0,zero
3,-5,minus
4,0,zero
```
### 4.50 int - Integer

Format: int(num)

Return the integer value of num. Plus and minus signs are returned as is.

#### Example

**Example 1: Basic Example**

```bash
$ more dat1.csv
id,val
1,3.14
2,3
3,12.56789
4,1.2345e+2
5,1.2345e-10
$ mcal c='int(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=int(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.14,3
2,3,3
3,12.56789,-12
4,1.2345e+2,123
5,1.2345e-10,0
```
4.51 match - Search

Format 1: match(search string ,str1, str2, ⋯)

Format 2: matcha(search string ,str1, str2, ⋯)

Format 3: matches(search string ,str1, str2, ⋯)

Format 4: matchas(search string ,str1, str2, ⋯)

Search for the search string in str1, str2, ⋯, and returns true if there is a match and false otherwise.

OR search and AND search returns partial match and exact match of string. The corresponding functions are shown in Table 4.22.

<table>
<thead>
<tr>
<th>Input Data</th>
<th>OR search</th>
<th>AND search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact match</td>
<td>match</td>
<td>matcha</td>
</tr>
<tr>
<td>Partial match</td>
<td>matches</td>
<td>matchas</td>
</tr>
</tbody>
</table>

The match function returns true if there is an exact match of the specified character string with any of the string in str1, str2, ⋯. The matcha function returns true if there is an exact match with all strings in str1, str2, ⋯. matches function returns true if there is partial match with any of the string in str1, str2, ⋯. matchas function returns true if there is partial match with all strings in str1, str2, ⋯. Refer to Table 4.8 on the boolean table of OR/AND logical operation for NULL values.

Examples

Example 1: OR exact match

Returns true if either column f1, f2, f3 contains 1.

```
$ more dat1.csv
id,f1,f2,f3
1,1,1,1
2,1,0,1
3,_,_,1
4,1,1,1
$ mcal c='match("1",$s{f1},$s{f2},$s{f3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=match("1",$s{f1},$s{f2},$s{f3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,f1,f2,f3,rsl
1,1,1,1,1
2,1,0,1,1
3,_,_,0,0
4,1,1,1,1
```

Example 2: AND exact match

Returns true if columns f1, f2, f3 contains the character "1".

```
$ mcal c='matcha("1",$s{f1},$s{f1},$s{f1},$s{f1},$s{f1})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=matcha("1",$s{f1},$s{f1},$s{f1},$s{f1},$s{f1}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,f1,f2,f3,rsl
1,1,1,1,1
2,1,0,1,1
3,_,_,0,0
4,1,1,1,1
```

Example 3: OR partial match

Returns true if the character string ab exists in either column s1, s2, s3.

```
$ mcal c='match("ab",$s{s1},$s{s2},$s{s3})' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=match("ab",$s{s1},$s{s2},$s{s3}) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,s1,s2,s3,rsl
1,1,1,1,1
2,1,0,1,1
3,_,_,0,0
4,1,1,1,1
```
Example 4: AND partial match

Returns true if the character string \texttt{ab} exists in columns \texttt{s1}, \texttt{s2}, \texttt{s3}.

Example 5: Search for NULL value

Return true if \texttt{str} column contains NULL value.
4.52. **ISNULL - EVALUATE NULL VALUE**

**4.52 isnull - Evaluate NULL value**

Format: isnull(num), isnull(str), isnull(date), isnull(time), isnull(bool)

Determine whether NULL value is included in the value `num` (the same applies to other data type). Returns 0b1 (true) if NULL value exists, and 0b0 (false) otherwise.

**Examples**

**Example 1: Basic Example**

```
$ more dat1.csv
id,val
1,a
2,
3,b
$ mcal c='isnull(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=isnull(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,a,0
2,,1
3,b,0
```

**Example 2: Specify other field types**

```
$ mcal c='isnull($s{val})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=isnull($s{val}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,a,0
2,,1
3,b,0
```

**Example 3: Specify null character**

```
$ mcal c='isnull("")' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=isnull("") i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,val,rsl
1,a,1
2,,1
3,b,1
```
4.53   juli an - Julian Calendar Conversion

Format 1: julian\texttt{(date)}
Format 2: julian\texttt{(time)}
Format 3: julian\texttt{2d(num)}
Format 4: julian\texttt{2t(num)}

Format 1 and 2 converts \texttt{date} or \texttt{time} to Julian day of year. Format 3 and 4 reverse the conversion from Julian day of year to date or time. Given the date type, the beginning of the day is counted as \texttt{00:00:00}.

Examples

Example 1: Basic Example

Convert the data in the \texttt{date} column to Julian day of year with \texttt{d2julian} formula, and convert back using \texttt{julian2d} function.

```bash
$ more dat1.csv
id,date
1,20000101
2,20121021
3,4,19700101
$ mcal c='julian($d{date})' a=julian i=dat1.csv o=rsl1.csv
#END# kgcal a=julian c=julian($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,julian
1,20000101,2451545
2,20121021,2456222
3,,
4,19700101,2440588
$ mcal c='julian2d(${julian})' a=date2 i=rsl1.csv o=rsl2.csv
#END# kgcal a=date2 c=julian2d(${julian}) i=rsl1.csv o=rsl2.csv
$ more rsl2.csv
id,date,julian,date2
1,20000101,2451545,20000101
2,20121021,2456222,20121021
3,,
4,19700101,2440588,19700101
```

Example 2: Apply the function to time formatted data

```bash
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,4,19700101000100
$ mcal c='julian($t{time})' a=julian i=dat2.csv o=rsl3.csv
#END# kgcal a=julian c=julian($t{time}) i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,time,julian
1,20000101000000,2451545
2,20121021111213,2456222.467
3,,
4,19700101000100,2440588.001
$ mcal c='julian2t(${julian})' a=time2 i=rsl3.csv o=rsl4.csv
#END# kgcal a=time2 c=julian2t(${julian}) i=rsl3.csv o=rsl4.csv
$ more rsl4.csv
id,time,julian,time2
1,20000101000000,2451545,20000101000000
2,20121021111213,2456222.467,20121021000000
3,,
4,19700101000100,2440588.001,19700101000000
```
4.54 lcm - Least Common Denominator

Format: lcm(num1, num2)

Find out the least common denominator between num1 and num2. Real number will be rounded down to whole number.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,val1,val2
1,5,3
2,12,4
3,4,5.1,3.1
$ mcal c='lcm(${val1},${val2})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=lcm(${val1},${val2}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val1,val2,rsl
1,5,3,15
2,12,4,15
3,4,5.1,3.1,15
```

Example 2: Example using constant

Find out the least common denominator between column val1 and 15.

```
$ mcal c='lcm(${val1},15)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=lcm(${val1},15) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val1,val2,rsl
1,5,3,15
2,12,4,60
3,4,5.1,3.1,15
```
4.55  leapyear - Determine Leap Year

Format 1: leapyear(date)

Format 2: leapyear(time)

Determine leap year from date or time.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,date
1,20000101
2,20121021
3,4,19770812
$ mcal c='leapyear($d{date})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=leapyear($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,1
2,20121021,1
3,,
4,19770812,0
```

Example 2: Determine leap year from time formatted data

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,4,19770812122212
$ mcal c='leapyear($t{time})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=leapyear($t{time}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,1
2,20121021111213,1
3,,
4,19770812122212,0
```
4.56 left - Extract String from Left

Format 1: left(str, length)

Format 2: leftw(str, length)

Extract specified number of characters from the left as defined in the length parameter from the character string str. Use leftw if the string contains multibyte characters.

Examples

Example 1: Basic Example

Extract the first 3 characters in the str column.

```
$ more dat1.csv
id,str
1,abcdefg
2,12345678
3,
4,12
$ mcal c='left($s{str},3)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=left($s{str},3) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abcdefg,abc
2,12345678,123
3,
4,12,12
```
4.57 length - Length of Character String

Format 1: length(str)

Format 2: lengthw(str)

Compute the length of character string. Use lengthw function to initialize wide character in str. Note that length of NULL value is 0.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,str
1, abc
2,3.1415
3, hello world!

$ mcal c='length($s{str})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=length($s{str}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1, abc, 3
2,3.1415, 6
3, hello world!, 12
```

Example 2: Include multibyte character

The following example uses Japanese in UTF-8 encoding. Each UTF-8 Japanese character is encoded in 3 bytes, thus, the length function returns the number of bytes rather than the number of Japanese characters.

```
$ more dat2.csv
id,str
1, こんにちは
2, 大阪

$ mcal c='length($s{str})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=length($s{str}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1, こんにちは, 15
2, 大阪, 6
```

Example 3: Initialize wide character

The lengthw function converts each wide characters internally into multibyte character for computation.

```
$ mcal c='lengthw($s{str})' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=lengthw($s{str}) i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,str,rsl
1, こんにちは, 15
2, 大阪, 6
```
4.58  line - Line Number

Format: line()

Return the line number processed by mcal command. mcmd standarize all line numbers starting from 0, similarly, line function initializes the first row of data from 0.

Examples

Example 1: Basic Example

Print number starting from 0 in output.

```
$ more dat1.csv
id
1
2
3
4
$ mcal c='line()' a=no i=dat1.csv o=rsl1.csv
#END# kgcal a=no c=line() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,no
1,0
2,1
3,2
4,3
```

Example 2: Start from 1

Print number starting from 1 in output.

```
$ mcal c='line()+1' a=no i=dat1.csv o=rsl2.csv
#END# kgcal a=no c=line()+1 i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,no
1,1
2,2
3,3
4,4
```
4.59 \textbf{ln - Natural Logarithm}

Format: \texttt{ln(num,base)}

Compute the natural logarithm of \textit{num}.

\textbf{Example}

\textbf{Example 1: Basic Example}

```
$ more dat1.csv
id,val
1,10
2,2.718281828
3,4.0
4,1
5,-8
$ mcal c='ln(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=ln(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,10,2.302585093
2,2.718281828,0.9999999998
3,4.0,1.0
4,1.0
5,-8,
```
4.60  log - Logarithm

Format: \text{log}(\text{num},\text{base})

Compute logarithm of \text{num} with specified \text{base}.

Example

Example 1: Basic Example

\begin{verbatim}
$ more dat1.csv
id,val,base
1,100,10
2,256,2
3,,
4,2,0
5,0,2
6,1,10
7,-8,2
$ mcal c='log(${val},${base})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=log(${val},${base}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,base,rsl
1,100,10,2
2,256,2,8
3,,
4,2,0,-0
5,0,2,
6,1,10,0
7,-8,2,
\end{verbatim}
4.61 log10 - Common logarithm

Format: \( \log_{10}(num, base) \)
Compute common logarithm of \( num \).

Example

Example 1: Basic Example

```bash
$ more dat1.csv
id,val
1,10
2,0.1
3,
4,0
5,1
6,-8
$ mcal c='log10(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=log10(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,10,1
2,0.1,-1
3,
4,0,
5,1,0
6,-8,
```
4.62  log2 - Log base 2

Format: log2(num)

Compute logarithm of num with base 2.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,10
2,256
3,4
5,1
6,-8
$ mcal c='log2(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=log2(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,10,3.321928095
2,256,8
3,4,0
5,1,0
6,-8,
```
4.63 match - Search

Format 1: match(search string ,str1, str2, ⋯)
Format 2: matcha(search string ,str1, str2, ⋯)
Format 3: matches(search string ,str1, str2, ⋯)
Format 4: matchas(search string ,str1, str2, ⋯)

Search for the search string in str1, str2, ⋯, and returns true if there is a match and false otherwise.

OR search and AND search returns partial match and exact match of string. The corresponding functions are shown in Table 4.22.

<table>
<thead>
<tr>
<th>OR search</th>
<th>AND search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact match</td>
<td>match</td>
</tr>
<tr>
<td>Partial match</td>
<td>matches</td>
</tr>
</tbody>
</table>

The match function returns true if there is an exact match of the specified character string with any of the string in str1, str2, ⋯. The matcha function returns true if there is an exact match with all strings in str1, str2, ⋯. The matches function returns true if there is partial match with any of the string in str1, str2, ⋯. The matchas function returns true if there is partial match with all strings in str1, str2, ⋯. Refer to Table 4.8 on the boolean table of OR/AND logical operation for NULL values.

Examples

Example 1: OR exact match

Returns true if either column f1, f2, f3 contains 1.

```
$ more dat1.csv
id,f1,f2,f3
1,1,1,1
2,1,0,1
3,,,
4,1,1
$ mcal c='match("1",$s{f1},$s{f2},$s{f3})' a=rsl i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,f1,f2,f3,rsl
1,1,1,1,1
2,1,0,1,1
3,,0,0
4,1,1,1
```

Example 2: AND exact match

Returns true if columns f1, f2, f3 contains the character "1".

```
$ mcal c='matcha("1",$s{f1},$s{f2},$s{f3})' a=rsl i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,f1,f2,f3,rsl
1,1,1,1,1
2,1,0,1,0
3,,0,0
4,1,1,1
```

Example 3: OR partial match

Returns true if the character string ab exists in either column s1, s2, s3.

```
$ mcal c='match("ab",$s{s1},$s{s2},$s{s3})' a=rsl i=dat1.csv o=rsl3.csv
```

Examples
Example 4: AND partial match

Returns true if the character string ab exists in columns s1, s2, s3.

```bash
$ mcal c='matchas("ab",$s{s1},$s{s2},$s{s3})' a=rsl i=dat2.csv o=rsl4.csv
#END# kgcal a=rsl c=matchas("ab",$s{s1},$s{s2},$s{s3}) i=dat2.csv o=rsl4.csv
$ more rsl4.csv
id,s1,s2,s3,rsl
1,ab,abx,x,0
2,abc,xaby,xxab,1
3,,,,0
4,#ac,x,x,0
```

Example 5: Search for NULL value

Return true if str column contains NULL value.

```bash
$ mcal c='match(nulls(),$s{s1},$s{s2},$s{s3})' a=rsl i=dat2.csv o=rsl5.csv
#END# kgcal a=rsl c=match(nulls(),$s{s1},$s{s2},$s{s3}) i=dat2.csv o=rsl5.csv
$ more rsl5.csv
id,s1,s2,s3,rsl
1,ab,abx,x,0
2,abc,xaby,xxab,0
3,,,,1
4,#ac,x,x,0
```
4.64 max - Maximum Value

Format: max(num1,num2,...)

Compute the maximum number in num. NULL values are ignored and returned as NULL value.

Examples

Example 1: Basic Example

```bash
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,...
$ mcal c='max(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=max(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,3
2,-5,2,1,2
3,1,,3,3
4,...
```

Example 2: Example using wildcard

Specify columns starting from v (v1,v2,v3) using wildcard \[ v* \].

```bash
$ mcal c='max(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=max(${v*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,3
2,-5,2,1,2
3,1,,3,3
4,...
```
4.65 mid - Extract Substring

Format 1: mid(str, starting position, length)
Format 2: midw(str, starting position, length)

Extract the character string str from the specified starting position for a specified length. Note that the starting position starts from 0. Use leftw function if the string contains multibyte characters.

Example

Example 1: Basic Example

Extract the first 3 characters from the beginning in the str column.

```
$ more dat1.csv
id,str
1,abcdefg
2,12345678
3,
4,12
$ mcal c='mid($s{str},2,3)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=mid($s{str},2,3) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abcdefg,cde
2,12345678,345
3,12
4,12,
```
CHAPTER 4. MCAL

4.66  min - Minimum Value

Format: min(num1, num2, \ldots)

Compute the minimum number in numi. NULL values are ignored and returned as NULL value.

Examples

Example 1: Basic Example

```bash
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,

$ mcal c='min(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=min(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv

$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,1
2,-5,2,1,-5
3,1,,3,1
4,,
```

Example 2: Example using wildcard

Use wildcard \( v^* \) to specify columns starting from \( v \) (v1, v2, v3).

```bash
$ mcal c='min(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=min(${v*}) i=dat1.csv o=rsl2.csv

$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,1
2,-5,2,1,-5
3,1,,3,1
4,,
```
4.67 minute - Minute

Format 1: minute(time) Numerical value
Format 2: minutes(time) 2 digit fixed length string
Extract minute from time. The output is returned in a format listed in format 1 and 2.

Example

Example 1: Basic Example

```bash
$ more dat1.csv
id,time
1,20000101000000
2,20120211111213
3,
4,19770812122212
$ mcal c='minute($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=minute($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,20000101000000,0
2,20120211111213,12
3,
4,19770812122212,22
```

Example 2: Print in character string

```bash
$ mcal c='minutes($t{time})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=minutes($t{time}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,0
2,20120211111213,12
3,
4,19770812122212,22
```
4.68  month - Month

Format 1: month(dt)  Numeric month

Format 2: months(dt)  Two digit fixed length numeric month in character string

Format 3: monthe(dt)  Month in English

Format 4: monthes(dt)  Abbreviation of month in English

Return month from date and time data. Month can be expressed in different formats as shown in format 1 to 4.

Example

Example 1: Basic Example

```c
$ more dat1.csv
id,date
1,20000101
2,20121021
3,4,19770812

$ mcal c='month($d{date})' a=rsl i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,1
2,20121021,10
3,4,19770812,8
```

Example 2: Fixed length character string

```c
$ mcal c='months($d{date})' a=rsl i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,date,rsl
1,20000101,01
2,20121021,10
3,4,19770812,08
```

Example 3: Month in English

```c
$ mcal c='monthe($d{date})' a=rsl i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,date,rsl
1,20000101,January
2,20121021,October
3,4,19770812,August
```

Example 4: Abbreviation of month in English

```c
$ mcal c='monthes($d{date})' a=rsl i=dat1.csv o=rsl4.csv
$ more rsl4.csv
id,date,rsl
1,20000101,Jan
2,20121021,Oct
3,4,19770812,Aug
```
Example 5: Extract month from time formatted data

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
4,19770812122212
$ mcal c='month($t{time})' a=rsl i=dat2.csv o=rsl5.csv
$ more rsl5.csv
id,time,rsl
1,20000101000000,1
2,20121021111213,10
4,19770812122212,8
```
4.69  not - Negation

Format: \texttt{not(bool)}

Returns the reversed logical value of \texttt{bool}.

Example

Example 1: Basic Example

\begin{verbatim}
$ more dat1.csv
id,b
1,1
2,
3,0
$ mcal c='not($b{b}())' a=rsl i=dat1.csv o=rsl1.csv
END# kgcal a=rsl c=not($b(b)) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,b,rsl
1,1,0
2,\n3,0,1
\end{verbatim}
4.70  **now - Current Time**

Format: `now()`

The `now` function returns current time in time format.

**Example**

**Example 1: Basic Example**

```bash
$ more dat1.csv
id
1
2
$ mcal c='now()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=now() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,20131102132754
2,20131102132754
```

**Example 2: Extract time from time formatted data**

```bash
$ mcal c='time(now())' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=time(now()) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,rsl
1,132754
2,132754
```

**Example 3: Extract date from time formatted data**

```bash
$ mcal c='date(now())' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=date(now()) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,rsl
1,20131102
2,20131102
```
4.71 null - NULL Value

Format: nulln(),nulls(),nullid(),nullt(),nullb()

Return NULL value for corresponding types. It can be used to return NULL value in conjunction with the if function.

Example

Example 1: Basic Example

Print NULL values to column rsl.

```bash
$ more dat1.csv
id
1
2
3
$mcal c='nulls()' a=rsl i=dat1.csv o=rsl1.csv
$END# kgcal a=rsl c=nulls() i=dat1.csv o=rsl1.csv
$m more rsl1.csv
id,rsl
1,
2,
3,
```

Example 2: Use of if statement

Use nulln() function to match the value specified in the second parameter.

```bash
$mcal c='if(${id}==1,1,nulln())' a=rsl i=dat1.csv o=rsl2.csv
$END# kgcal a=rsl c=if(${id}==1,1,nulln()) i=dat1.csv o=rsl2.csv
$m more rsl2.csv
id,rsl
1,1
2,
3,
```

Example 3: Equivalent to isnull function

```bash
$mcal c='if(${val}==nulln(),"null","notNull")' a=rsl i=dat2.csv o=rsl3.csv
$END# kgcal a=rsl c=if(${val}==nulln(),"null","notNull") i=dat2.csv o=rsl3.csv
$m more rsl3.csv
id,val,rsl
1,a,
2,.
3,b,
```
4.72 nrand - Random Numbers in Normal Distribution

Generate random numbers with specified mean and standard deviation.

The same random seed will generate the same sequence of random numbers. The random seed can be specified from -2147483648 to 2147483647. If the random seed is not defined, the random numbers will be seeded according to current time (1/1000 second).

This function uses Mersenne Twister as random number generator (Author’s original website, boost library)

Example

Example 1: Basic Example

Create a set of random numbers with a standard deviation between 0 to 1 (standard normal distribution). Set the random seed as 10.

```bash
$ more dat1.csv
id
1
2
3
4

$ mcal c='nrand(0,1,10)' a=rsl i=dat1.csv o=rsl1.csv
#END# kcal a=rsl c=nrand(0,1,10) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,0.1125259159
2,-0.8349559194
3,1.158317265
4,0.1518928088
```
4.73 or - Logical Disjunction

Format: or(bool₁, bool₂,...)

Compute the boolean result using logical disjunction on the array of boolean values in boolᵢ. Refer to Table 4.9 on the boolean truth table with NULL values.

Examples

Example 1: Basic Example

```bash
$ more dat1.csv
id,b1,b2,b3
1,1,0,0
2,1,1
3,0,0,0
4,0,0,0

$ mcal c='or($b{b1},$b{b2},$b{b3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=or($b{b1},$b{b2},$b{b3}) i=dat1.csv o=rsl1.csv

$ more rsl1.csv
id,b1,b2,b3,rsl
1,1,0,0,1
2,1,1,1
3,0,0,0
4,0,0,0

Example 2: Example using wildcard

Use the wildcard character □ b* □ to specify columns starting from b (b₁,b₂,b₃).

```bash
$ mcal c='or($b{b*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=or($b{b*}) i=dat1.csv o=rsl2.csv

$ more rsl2.csv
id,b1,b2,b3,rsl
1,1,0,0,1
2,1,1,1
3,0,0,0
4,0,0,0
```
4.74  pi - Pi

Format: pi()

This function returns the value of Pi (\(\pi\)).

Example

Example 1: Basic Example

```
$ more dat1.csv
id
1
2
$ mcac c='pi()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=pi() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,3.141592654
2,3.141592654
```
4.75  power - Raised to a Power

Format: power(num,power)

Calculate the power of num. This function is equivalent to the \(^\) operator. The function returns NULL value if the result exceeds the largest real number.

Example

Example 1: Basic Example

```bash
$ more dat1.csv
id,base,exponent
1,5,2
2,2,8
3,,
4,0,10
5,10,0
6,2,0.5
7,2,-1
$ mcal c='power(${base},${exponent})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=power(${base},${exponent}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,base,exponent,rsl
1,5,2,25
2,2,8,256
3,,
4,0,10,0
5,10,0,1
6,2,0.5,1.414213562
7,2,-1,0.5
```
4.76 product - Product

Format: product(num1, num2, ⋯)

Calculate the product of the array of values in num1. NULL values are ignored, if NULL values exist in all items, the result will return a NULL value.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,,;

$mcal c='product(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=product(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,6
2,-5,2,1,-10
3,1,,3,3
4,,,;
```

Example 2: Example using wildcard

Use the wildcard character □ v* □ to specify columns starting from v (v1,v2,v3).

```
$ mcald c='product(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=product(${v*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,6
2,-5,2,1,-10
3,1,,3,3
4,,,;
```
4.77  radian - Radian

Convert degrees to radians.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,60
2,180
3,4,0
5,-90

$ mcal c='radian(${val})' a=rsl i=dat1.csv o=rsl1.csv

$ more rsl1.csv
id,val,rsl
1,60,1.047197551
2,180,3.141592654
3,4,0
5,-90,-1.570796327
```
4.78  rand - Uniformly distributed random numbers

Format: ([Random seed])

Generate uniformly distributed random number from 0.0 to 1.0. This function returns the same result as using mrand command without -int option. The same random seed will generate the same sequence of random numbers. The random seed can be specified from -2147483648 to 2147483647. If the random seed is not defined, the random numbers will be seeded according to current time (1/1000 second). This function uses Mersenne Twister as random number generator (Author’s original website, boost library).

Examples

Example 1: Basic Example

Generate uniformly distributed random number from 0.0 to 1.0. Since a random seed is specified, the random number sequence generated is always the same.

```bash
$ more dat1.csv
id
1
2
3
4
$ mcal c='rand(1)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=rand(1) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,0.4170219984
2,0.9971848081
3,0.7203244893
4,0.9325573612
```

Example 2: Random seed is not set

The random sequence will be different upon each run since random seed is not specified.

```bash
$ mcal c='rand()' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=rand() i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,rsl
1,0.4007082446
2,0.8031022043
3,0.5551679884
4,0.9325573612
```
4.79  randi - Uniformly Generated Random Integers

Format: (minimum, maximum[, random seed])

Generate random integers between the specified minimum value to maximum value. The mrand command using -int option returns the same results. The same random seed will generate the same sequence of random numbers. The random seed can be specified from -2147483648 to 2147483647. If the random seed is not defined, the random numbers will be seeded according to current time (1/1000 second). This function uses Mersenne Twister as random number generator (Author’s original website, boost library).

Examples

Example 1: Basic Example

Generate random integers from 100 to 999 (3 digit integers 900 types). Since a random seed is specified, the random number sequence generated is always the same.

```
$ more dat1.csv
id
1
2
3
4
$ mcal c='randi(100,999,1)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=randi(100,999,1) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,475
2,997
3,748
4,939
```

Example 2: 0,1 random integers

Generate two sets of random integers 0 and 1. The random sequence will be different upon each run since the random seed is not specified.

```
$ mcal c='randi(0,1)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=randi(0,1) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,rsl
1,1
2,0
3,0
4,1
```
4.80  regexlen - Match Length of Character String

Format 1: `regexlen(str, regular expression)`

Format 2: `regexlenw(str, regular expression)`

Returns the length of substring where the defined regular expression matches the string `str`. The function returns 0 if no match is found, in other words, 0 character match.

Use `regexlenw` function if `str` or regular expression contain multibyte characters.

Examples

Example 1: Basic Example

Find out the length of the longest substring that matches with the regular expression `c.*a`. Since the same input data is used for matching substring in the `regexstr` function, it is easier to compare the results.

```
$ more dat1.csv
id,str
1,xcbbbayy
2,xxcbaay
3,
4,bacabbca
$ mcal c='regexlen($s{str},"c.*a")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexlen($s{str},"c.*a") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,xcbbbayy,5
2,xxcbaay,4
3,,0
4,bacabbca,6
```

Example 2: Multibyte characters

Find out the length of the longest substring that matches "い.*あ". However, since `regexlen` function do not support multibyte character, it returns the number of bytes instead of number of characters.

```
$ more dat2.csv
id,str
1,ぱぱあ bbb い yy
2,ぱ a い り yy
3,
4,b あ い a bb い a
$ mcal c='regexlen($s{str},"あ.*い")' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=regexlen($s{str},"あ.*い") i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,ぱぱあ bbb い yy,9
2,ぱ a い り yy,10
3,,0
4,b あ い a bb い a,14
```

Example 3: Multibyte characters 2

Find out the length of the longest substring that matches "い.*あ". The `regexlenw` function is able to process multibyte characters to count the number of characters.

```
$ mcal c='regexlenw($s{str},"あ.*い")' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=regexlenw($s{str},"あ.*い") i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,str,rsl
1,ぱぱあ bbb い yy,9
2,ぱ a い り yy,10
```
4.81 regexm - Complete Match

Format 1: regexm(str, regular expression)

Format 2: regexmw(str, regular expression)

Return true if the regular expressions matches the whole character string $str$. Use regexmw function if multibyte characters exists in $str$ or regular expression, or characters in Shift_JIS encoding does not correspond to search results.

Examples

Example 1: Basic Example

Both records where id=1, id=2 contains regular expression beginning with c and ends with aa, in record id=2, there is only partial match (matches from c at the second position to the end) with the regular expression.

```
$ more dat1.csv
id,str
1,caabaa
2,acabaaa
3,
4,bbcbcc
```

```
$ mcal c='regexm($s{str},"c.*aa")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexm($s{str},"c.*aa") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,caabaa,1
2,acabaaa,0
3,,0
4,bbcbcc,0
```

Example 2: String ends with same substring

The full string in column $str$ that matches the regular expression .*c is only found in the record where id=4.

```
$ mcal c='regexm($s{str},".*c")' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=regexm($s{str},".*c") i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,caabaa,0
2,acabaaa,0
3,,0
4,bbcbcc,1
```

Example 3: Match blank characters

Match the blank characters at id=3 using the regular expression `^$`.

```
$ mcal c='regexm($s{str},"^\$")' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=regexm($s{str},"^\$") i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,str,rsl
1,caabaa,0
2,acabaaa,0
3,,1
4,bbcbcc,0
```
4.82 regexpfx - Prefix of Match String

Format 1: regexpfx(str, regular expression)
Format 2: regexpfx(str, regular expression)

Return the prefix (substring before character string) of the longest substring where the defined regular expression matches with the character string str. When the same character string and regular expression is used with the three functions to extract different parts of the string in sequential order, namely regexpfx function, regexstr function, and regexsfx function, the original string can be restored by merging the resulting character strings from the functions. Use the regexpfxw function if str or regular expression contain multibyte characters, or when characters in Shift_JIS encoding does not correspond to search results.

Example

Example 1: Basic Example

Find out the prefix of the longest substring that matches with the regular expression c.*a. For example, in record id=4, the function matches cabbca in the column and returns the prefix ba. Since the same input data and substring are used in the regexstr, regexpfx function, it is easy to compare the results.

```
$ more dat1.csv
id,str
1,xcbbbayy
2,xxcbaay
3,4,bacabbca
$ mcal c='regexpfx($s{str},"c.*a")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexpfx($s{str},"c.*a") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,xcbbbayy,x
2,xxcbaay,xx
3,4,bacabbca,ba
```
4.83 regexpos - Match Position

Format 1: regexpos(str,regular expression)

Format 2: regexposw(str,regular expression)

Return the starting position of the longest substring where the defined regular expression matches with the character string str. The first character of the string starts at position 0. The function return NULL if no match is found. Use the regexposw function if str or regular expression contain multibyte characters.

Examples

Example 1: Basic Example

Find out the position of the longest substring that matches the regular expression c.*a. Note that the first character starts at position 0. Since the same input data and substring are used in the regexstr function, it is easy to compare the results and understand the differences.

```
$ more dat1.csv
id,str
1,xcbbbayy
2,xxcbaay
3,4,bacabbca
$ mcal c='regexpos($s{str},"c.*a")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexpos($s{str},"c.*a") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,xcbbbayy,1
2,xxcbaay,2
3,4,bacabbca,2
```

Example 2: Multibyte character

Find out the longest substring that matches the regular expression "あ.*あ". However, since regexpos does not support multibyte characters, the function returns the position of bytes instead of characters.

```
$ more dat2.csv
id,str
1,漢漢あ bbbい yy
2,漢あ bいい い y
3,4,あああ b bい い a
$ mcal c='regexpos($s{str},"あ.*あ")' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=regexpos($s{str},"あ.*あ") i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,漢漢あ bbbい yy,6
2,漢あ bいい い y,3
3,4,あああ b bい い a,1
```

Example 3: Multibyte character 2

Find out the longest substring that matches the regular expression "あ.*あ". The regexposw function is able to process multibyte characters accurately.

```
$ mcal c='regexposw($s{str},"あ.*あ")' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=regexposw($s{str},"あ.*あ") i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,str,rsl
1,漢漢あ bbbい yy,6
2,漢あ bいい い y,3
```
3.
4. b あいあ いいあ、1
### 4.84 regexrep - Replace Match String

Format 1: `regexrep(str, regular expression, replacement string)`
Format 2: `regexrepw(str, regular expression, replacement string)`

Return the starting position of the longest substring where the defined regular expression matches with the character string `str`. Use the `regexrepw` function if `str` or regular expression contain multibyte characters, or when characters in Shift_JIS encoding does not correspond to search results.

**Example**

**Example 1: Basic Example**

In row where `id=1,id=2`, the matched substrings in the column is replaced with `MMM`.

```
$ more dat1.csv
id,str
1,caabaa
2,acabaaa
3,
4,cbcbcc
$ mcal c='regexrep($s{str},"c.*aa","MMM")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexrep($s{str},"c.*aa","MMM") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,caabaa,MMM
2,acabaaa,aMMM
3,
4,cbcbcc,cbcbbc
```
4.85 regexs - Substring Match

Format 1: regexs(str, regular expression)

Format 2: regexsw(str, regular expression)

Return the starting position of the longest substring where the defined regular expression matches with part of the character string str. Use the regexsw function if str or regular expression contain multibyte characters, or when characters in Shift_JIS encoding does not correspond to search results.

Example

Example 1: Basic Example

Records that contain the regular expression starting with c until aa includes id=1, id=2. The function returns true for matching records.

```bash
$ more dat1.csv
id,str
1,caabaa
2,acabaaa
3,
4,cbcbcc
$ mcal c='regexs($s{str},"c.*aa")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexs($s{str},"c.*aa") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,caabaa,1
2,acabaaa,1
3,,0
4,cbcbcc,0
```

Example 2: Match start of string

The regular expression .*c which matches the value in the str column for all records except id=3.

```bash
$ mcal c='regexs($s{str},".*c")' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=regexs($s{str},".*c") i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,caabaa,1
2,acabaaa,1
3,,0
4,cbcbcc,1
```
4.86  regexsfx - Suffix of Matched String

Format 1: regexsfx(str, regular expression)
Format 2: regexsfx(str, regular expression)

Return the suffix (substring at the end of character string) of the longest substring where the defined regular expression matches part of the character string str. When the same character string and regular expression is used with the three functions to extract different parts of the string in sequential order, namely regexpfx function, regexstr function, and regexsfx function, the original string can be restored by merging the resulting character strings from the functions. Use the regexsfxw function if str or regular expression contain multibyte characters, or when characters in Shift-JIS encoding does not correspond to search results.

Example

Example 1: Basic Example

Extract the suffix of the longest substring that matches the regular expression c.*a. For example, in row where id=4, the regular expression matches cabbca till the ending position in the column, there is no suffix to the matching string, thus NULL character is returned. Since the same input data and substring are used as in the regexstr, regexpfx function, it is easy to compare the results and understand the differences.

```
$ more dat1.csv
id,str
1,xcbbbayy
2,xxcbaay
3,
4,bacabbca
$ mcal c='regexsfx($s{str},"c.*a")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexsfx($s{str},"c.*a") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,xcbbbayy,yy
2,xxcbaay,y
3,,
4,bacabbca,
```
4.87 regexstr - Match Character String

Format 1: regexstr(str, regular expression)

Format 2: regexstrw(str, regular expression)

Return the longest substring where the defined regular expression matches part of the character string str. Use the regexstrw function if str or regular expression contain multibyte characters, or when characters in Shift_JIS encoding does not correspond to search results.

Example

Example 1: Basic Example

Extract the longest substring that matches the regular expression c.*a. At row where id=2, the regular expression matches the string cba and cbaa, however, the longer substring is returned.

```
$ more dat1.csv
id,str
1,xcbbbayy
2,xxcbaay
3,
4,bacabbca
$ mcal c='regexstr($s{str},"c.*a")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexstr($s{str},"c.*a") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,xcbbbayy,cbbba
2,xxcbaay,cbaa
3,
4,bacabbca,cabbca
```
### 4.88 right - Extract Substring from Right

Format 1: `right(str, length)`

Format 2: `rightw(str, length)`

Return the number of characters as defined in the parameter from the right side of the string in `str`. Use `rightw` function if the string contains multibyte characters.

#### Example

**Example 1: Basic Example**

Extract the last 3 characters from the end in the `str` column.

```bash
$ more dat1.csv
id,str
1,abcdefg
2,12345678
3,4,12
$ mcal c='right($s{str},3)' a=rsl i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abcdefg,efg
2,12345678,678
3,4,12
```

**Example 2: Example of data containing multibyte characters**

Use the function `right` if the data contains multibyte characters.

```bash
$ more dat2.csv
id,str
1,あいうえお
2,12345678あ8
3,あ
4,ああ
$ mcal c='right($s{str},6)' a=rsl i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,あいうえお,えお
2,12345678あ8,あ8
3,あ,あ
4,ああ,ああ
```
4.89  round - Rounding

Format: `round(num,base)`

Rounding `num` to the specified base. The number will be rounded to the nearest integer close to the multiple of the base. For example, in the expression `(3.82,0.5)`, the decimal digits of 3.82 is above 0.5, thus rounding to the nearest 0.5 returns 4.0. The default value of the base is 1 if the argument is not specified. This is equivalent to rounding an integer value to the first digit after decimal.

Example

Example 1: Basic Example

Round to the nearest digit after decimal.

```
$ more dat1.csv
id,val
1,3.28
2,3.82
3,4,-0.6
$ mcal c='round(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=round(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.28,3
2,3.82,4
3,4,-0.6,-1
```

Example 2: Basic Example

Round to the first decimal.

```
$ mcal c='round(${val},0.1)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=round(${val},0.1) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,3.28,3.3
2,3.82,3.8
3,4,-0.6,-0.6
```

Example 3: Example using base of 0.5

Round to the nearest 0.5.

```
$ mcal c='round(${val},0.5)' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=round(${val},0.5) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,val,rsl
1,3.28,3.5
2,3.82,4
3,4,-0.6,-0.5
```

Example 4: Example using base 10

Round to the nearest 10th digit.

```
$ more dat2.csv
id,val
1,1341.28
2,188
3,1.235E+3
```
4.89. ROUND - ROUNding

4.3E+3

$ mcal c='round({val},10)' a=rsl i=dat2.csv o=rsl4.csv

#END# kgcal a=rsl c=round({val},10) i=dat2.csv o=rsl4.csv

$ more rsl4.csv

id,val,rsl
1,1341.28,1340
2,188,190
3,1.235E+3,1240
4,-1.235E+3,-1230
4.90 second - Second

Format 1: second(time) numeric value
Format 2: seconds(time) 2 digit fixed length string

Extract second from time. The result of the function is noted in format 1 and 2.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,time
1,20000101000000
2,20121021111121
3,
4,19770812122212
$ mcal c='second($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=second($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,20000101000000,0
2,20121021111121,13
3,
4,19770812122212,12
```

Example 2: Return character string

```
$ mcal c='seconds($t{time})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=seconds($t{time}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,00
2,20121021111121,13
3,
4,19770812122212,12
```
4.91 sign - Sign

Format: sign(num)

Determine the sign of num. Returns 0 if zero, 1 if positive, -1 if negative.

Example

Example 1: Basic Example

```bash
$ more dat1.csv
id,val
1,5
2,-5
3,
4,0
$ mcal c='sign(${val})' a=rsl i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,5,1
2,-5,-1
3,
4,0,0
```
4.92  \( \sin - \text{Sine} \)

Format: \( \sin(r) \)
Calculate sine of radian \( r \).

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.141592
2,0.523599
3,4,6.283185
$ mcal c='sin(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=sin(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,6.535897931e-07
2,0.523599,0.5000001943
3,,4,6.283185,-3.071795869e-07
```
4.93 sinh - Hyperbolic Sine

Format: sinh(r)
Calculate sine of radian r.

Example

Example 1: Basic Example

```bash
$ more dat1.csv
id,val
1,3.141592
2,-1.047197
3,
4,6.283185
$ mcal c=' sinh(${val}) ' a=rsl i=dat1.csv o=rsl1.csv
#END# kgc a=rsl c=sinh(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,11.54873178
2,-1.047197,-1.249366168
3,
4,6.283185,267.7448118
```
4.94 \texttt{sqrt} - Square Root

Format: \texttt{sqrt(num)}

Find out the square root of \texttt{num}.

Example

Example 1: Basic Example

\begin{verbatim}
$ more dat1.csv
id,val
1,9
2,2
3,4
$ mcal c='sqrt(val)' a=rsl i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,9,3
2,2,1.414213562
3,4,-1
\end{verbatim}
4.95 sqsum - Sum of Squares

Format: sqsum(num1, num2, \ldots)

Calculate the sum of squares of the array of numbers in numi. NULL values are ignored, if NULL values exist in all numbers in the array, the result will return a NULL value.

Example

Example 1: Basic Example

```bash
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,
$ mcal c='sqsum(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=sqsum(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,14
2,-5,2,1,30
3,1,,3,10
4,,
```

Example 2: Example using wildcard

Specify fields starting with v (v1, v2, v3) by using wildcard "v*".

```bash
$ mcal c='sqsum(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=sqsum(${v*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,14
2,-5,2,1,30
3,1,,3,10
4,,
```
4.96  sum - Sum

Format: sum(num₁, num₂, ⋮)

Calculate the sum of the array of numbers in numᵢ. NULL values are ignored, if NULL values exist in all numbers in the array, the result will return a NULL value.

Example

Example 1: Basic Example

```bash
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,, $ mcal c='sum(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=sum(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,6
2,-5,2,1,-2
3,1,,3,4
4,,,#
```

Example 2: Example using wildcard

Specify fields starting with v (v1, v2, v3) by using wildcard "v*".

```bash
$ mcal c='sum(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=sum(${v*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,6
2,-5,2,1,-2
3,1,,3,4
4,,,#
```
4.97 TAN - TANGENT

4.97 tan - Tangent

Format: \( \tan(r) \)

Compute tangent of radian \( r \).

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,0.785398
2,1.047197
3,
4,3.141593
$ mcal c='tan(\${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=tan(\${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,0.785398,0.9999996732
2,1.047197,1.732048603
3,
4,3.141593,3.464102066e-07
```
4.98  \( \text{tanh} \) - Hyperbolic Tangent

Format: \( \text{tanh}(r) \)

Calculate the hyperbolic tangent.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.141592
2,-1.047197
3,4.6.283185
$ mcal c='tanh(${val})' a=rsl i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,0.9962720714
2,-1.047197,-0.7807142201
3,4.6.283185,0.9999930253
```
4.99 time - Hour Minute Second

Format: time(time)

Extract the 6-digit fixed length string of hour minute and second from time.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,time
1,20000101000000
2,20120211111213
3,
4,19770812122212
$ mcal c='time($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kcal a=rsl c=time($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,20000101000000,000000
2,20120211111213,111213
3,
4,19770812122212,122212
```
4.100 today - Today is date

Format: today()
Return today is date in date format.

Example

Example 1: Basic Example

```
$ more dat1.csv
id
1
2
$ mcal c='today()' a=rsl i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,20131102
2,20131102
```
4.101  **tolower - Lowercase Conversion**

Convert character string to lowercase characters. This does not affect non-alpha characters outside the 26 alphabets.

**Example**

**Example 1: Basic Example**

Convert the values in column `str` to lowercase characters.

```
$ more dat1.csv
id,str
1,ABC
2,aB$12!Cd
3,
4,cBA
$ mcal c='tolower($s{str})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=tolower($s{str}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,ABC,abc
2,aB$12!Cd,ab$12!cd
3,,
4,cBA,cba
```
4.102  top - Top Rows

Format: top()
Return true if first row, otherwise false.

Example

Example 1: Basic Example

```bash
$ more dat1.csv
val
1
2
3
4
$ mcal c='top()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=top() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
val,rsl
1,1
2,0
3,0
4,0
```

Example 2: Compute the cumulative value

```bash
$ mcal c='if(top(),${val},${val}+#{})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=if(top(),${val},${val}+#{}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
val,rsl
1,1
2,3
3,6
4,10
```


4.103 toupper - Uppercase Conversion

Convert character string to uppercase characters. This does not affect non-alpha characters outside the 26 alphabets.

Usage Examples

Example 1: Basic Example

Convert the values in str column from lowercase to uppercase.

```bash
$ more dat1.csv
id,str
1,abc
2,Ab$12!cD
3,
4,Cba
$ mcal c='toupper($s{str})' a=rsl i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abc,ABC
2,Ab$12!cD,AB$12!CD
3,,
4,Cba,CBA
```
4.104 tseconds - Elapsed Time in Seconds

Format: tseconds(time)
Compute the number of seconds elapsed from 00:00:00 to time

Usage Examples

Example 1: Basic Example

```sh
$ more dat1.csv
id,time
1,000103
2,235959
3,
4,000000
$ mcal c='tseconds($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=tseconds($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,000103,63
2,235959,86399
3,,
4,000000,0
```

Example 2: The result is the same using on date values

```sh
$ more dat2.csv
id,date
1,20130901000103
2,20130902000103
$ mcal c='tseconds($t{date})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=tseconds($t{date}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,date,rsl
1,20130901000103,63
2,20130902000103,63
```
4.105  

**uxt - UNIX Time Conversion**

Format 1: `uxt(date)`
Format 2: `uxt(time)`
Format 3: `uxt2d(num)`
Format 4: `uxt2t(num)`

In format 1 and 2, convert *date* or *time* to UNIX time. In 3 and 4 format, by contraries, convert UNIX time to date type or time type. Giving date type, calculate 00:00:00 as the first time of the day.

**Usage Examples**

**Example 1: Basic Example**

Convert the *date* formatted strings in the date column to UNIX time using `d2uxt` function, and convert back to original date string using `uxt2d` function.

```bash
$ more dat1.csv
id,date
1,20000101
2,20121021
3,
4,19700101
$ mcal c='uxt($d{date})' a=uxt i=dat1.csv o=rsl1.csv
#END# kgcal a=uxt c=uxt($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,uxt
1,20000101,946684800
2,20121021,1350777600
3,  
4,19700101,0
$ mcal c='uxt2d({uxt})' a=date2 i=rsl1.csv o=rsl2.csv
#END# kgcal a=date2 c=uxt2d({uxt}) i=rsl1.csv o=rsl2.csv
$ more rsl2.csv
id,date,uxt,date2
1,20000101,946684800,20000101
2,20121021,1350777600,20121021
3,  
4,19700101,0,19700101
```

**Example 2: Example of using time formatted data**

```bash
$ more dat2.csv
id,time
1,20000101000000
2,201210211111213
3,  
4,197001010000100
$ mcal c='uxt($t{time})' a=uxt i=dat2.csv o=rsl3.csv
#END# kgcal a=uxt c=uxt($t{time}) i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,time,uxt
1,20000101000000,946684800
2,201210211111213,1350817933
3,  
4,197001010000100,60
$ mcal c='uxt2t({uxt})' a=time2 i=rsl3.csv o=rsl4.csv
#END# kgcal a=time2 c=uxt2t({uxt}) i=rsl3.csv o=rsl4.csv
$ more rsl4.csv
id,time,uxt,time2
1,20000101000000,946684800,20000101000000
2,201210211111213,1350817933,201210211111213
3,  
4,197001010000100,60,197001010000100
```
### 4.106 Week

Format 1: `week(date)`

Format 2: `week(time)`

Format 3: `week111(date)`

Format 4: `week111(time)`

Return the week number following the ISO8601 standard from `date` or `time`. ISO8601 prescribed week number starts at the first week containing Thursday in the new ISO year. The function `week111` returns the week number that starts at the first day of week 01 on 1/1 regardless of the day of week.

**Usage Example**

**Example 1: Basic Example**

```
$ more dat1.csv
id,date
1,20000101
1,20000102
1,20000103
1,20000104
1,20000105
1,20000106
1,20000107
1,20000108
1,20000109
2,20121021
3,19770812
$ mcal c='week($d{date})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=week($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,52
1,20000102,52
1,20000103,1
1,20000104,1
1,20000105,1
1,20000106,1
1,20000107,1
1,20000108,1
2,20121021,42
3,19770812,32
```

**Example 2: Example of using time formatted data**

```
$ more dat2.csv
id,time
1,20000101000000
2,201210211111213
3,19770812122212
$ mcal c='week($t{time})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=week($t{time}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,52
2,201210211111213,42
3,19770812122212,32
```
4.107 year - Liturgical Year

Format 1: year(date)
Format 2: year(time)
Format 3: years(date)
Format 4: years(time)

Extract date and time from year. Format 1 and 2 return numerical value, format 3 and 4 return character string.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,date
1,20000101
2,20121021
3,4,19770812
$mcal c='year($d(date))' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=year($d(date)) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,2000
2,20121021,2012
3,4,19770812,1977
```

Example 2: Time formatted values

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,4,19770812122212
$mcal c='year($t(time))' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=year($t(time)) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,2000
2,20121021111213,2012
3,4,19770812122212,1977
```
4.108 Cast

Format: \text{s2n}(\text{str}), \text{n2s}(\text{num}), \text{n2b}(\text{num}), \text{s2n}(\text{str}), \text{s2d}(\text{str}), \text{s2t}(\text{str}) \text{s2b}(\text{str}), \text{d2s}(\text{date}), \text{d2t}(\text{date}), \text{t2s}(\text{time}), \text{t2d}(\text{time}), \text{b2n}(\text{bool}), \text{b2s}(\text{bool})

Set of functions to convert data type. \text{mcal} do not automatically converts data type, user must specify the required data type conversion.

\text{n2b(s2b)} function converts true to 1(“1”) and false to 0(“0”), other values are treated as NULL values.

\text{b2n(b2s)} function converts true to 1(“1”) and false to 0(“0”).

\text{d2t} function converts date and time type data, and automatically completes the time as 12:00:00. \text{d2s} function converts data to 8 digit fixed length character string (“yyyymmdd”), \text{t2s} function converts data to 14 digit fixed length character string (“yyyymmddHHMMSS”).

Refer "4.13 Date and Time Type" for more information on date and time.

Examples

Example 1: Fixed length random number

Generate random numbers from 1 to 9999 as 4 digit fixed length string. Integer data (results of randi) is not supported by \text{fixlen} function, thus the data must be converted to character string with \text{n2s} function.

```
$ more dat1.csv
id
1
2
3
4
$ mcal c='fixlen(n2s(randi(1,9999,11)),4,"R","0")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=fixlen(n2s(randi(1,9999,11)),4,"R","0") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,1803
2,0684
3,0195
4,6647
```

Example 2: True false pattern

Detect unusual pattern in columns v1,v2,v3 and print as 01 in output.

```
$ more dat2.csv
id,v1,v2,v3
1,10,5,7
2,5,12,11
3,3,6,2
4,14,16,11
$ mcal c='cat("",b2s(${v1}>=10),b2s(${v2}>=10),b2s(${v3}>=10))' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=cat("",b2s(${v1}>=10),b2s(${v2}>=10),b2s(${v3}>=10)) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3
1,10,5,7
2,5,12,11
3,3,6,2
4,14,16,11,011
```