MCMD Ruby Package Documentation

mcmdRB version: 1.0.0

Revision History:
November 2, 2013 : First Release

November 14, 2013

Contents

Chapter 1

Let's Start

1.1 Overview

This Ruby library contains powerful functions to handle large CSV data efficiently, and is comparably more efficient than similar libraries with the same functions. The main functions comprised of sequential read (Mcsvin) and write (Mcsvout) functions from CSV data, as well as random access to each cell within the CSV data table (Mtable). These functions treats newline and comma in strings according to the standard specification of CSV data compliant to RCF 4180. In addition, key fields can be used for Hash access to the data, and field numbers can represented as Array. Figure 1.1 below shows a sample script to copy CSV data using the functions Mcsvin and Mcsvout.

```
#!/usr/bin/env ruby
require "mcmd"

# input.csv
# customer,date
# A,20081201
# A,20081202
MCMD::Mcsvin.new("i=input.csv -array"){|iCSV|
  MCMD::Mcsvout.new("o=output.csv f=customer,date"){|oCSV|
    iCSV.each{|flds|
      oCsv.write(flds)
    }
  }
}
```

Figure 1.1: Ruby script to copy CSV file using mcmd library

Other than using mcmd library to copy CSV file using Ruby script, NYSOL has developed other commands using this library to be executed by Ruby script. This library provides the interface to basic functions similar to mcmd. Margs is used to process command line arguments, while the endMsg and errorMsg functions can be used to return status message or error message upon completion. The function Mtemp can be used to set the same environment variables as in mcmd such as generation and removal of temporary files.

This library is tested on Ruby version 1.9.2 or later.

1.2 Installation

M-Command supports the following operating system.

- Mac OS X 10.7.5(Lion)
- Ubuntu Linux 12.04(32bit, 64bit)

Installation packages are available for the above operating systems, users would be able to install MCMD on systems with slight variation of the OS versions listed above. The software can be compiled and installed from the source code for installation in other OS.

1.2.1 Mac OS X

Download the latest gem package from http://sourceforge.jp/projects/nysol/releases/ and install according to the following command. Note that in version number `"1.0"` in the file name "`mcmd-1.0.gem`" will be updated according to the latest version number.

```
$ wget http://sourceforge.net/xxxx
$ gem install mcmd-1.0_darwin.gem
```

1.2.2 Ubuntu Linux

Download the latest gem package from http://sourceforge.jp/projects/nysol/releases/ and install according to the following command. The file name follows the format illustrated below (`"1.1-0"` will be updated with the latest version).

- 32bit environment: `mcmd_1.1-0_i386.deb`
- 64bit environment: `mcmd_1.1-0_amd64.deb`

```
$ wget http://sourceforge.net/xxxx
$ gem install mcmd-1.0_darwin.gem
```

1.2.3 Install from Source

Follow the procedures below to compile and install the library from source code.

Install C++ boost library

Download and install[1] boost library from the C++ boost library website (`http://www.boost.org/`). For installation in Mac and Linux 32 OS, please follow the instructions as follows. It will take about 30 minutes to install boost library.

```
$ wget http://sourceforge.net/projects/boost/files/boost/1.52.0/boost_1_52_0.tar.gz/download
$ tar zxvf boost_1_52_0.tar.gz
$ cd boost_1_52_0
$ ./bootstrap.sh
$ ./bjam
$ sudo ./bjam install
```

Follow the step below for installation instructions on Linux 64bit OS environment.

```
$ wget http://sourceforge.net/projects/boost/files/boost/1.52.0/boost_1_52_0.tar.gz/download
$ tar zxvf boost_1_52_0.tar.gz
$ cd boost_1_52_0
$ ./bootstrap.sh
$ ./bjam cflags=-fPIC
$ sudo ./bjam install cflags=-fPIC
```

[1] This library is compatible with boost_1.52.0, however, this library cannot be compiled with version 1.54.0.

Install mcmd

Download and install the latest version of MCMD source code from the MCMD git server. Afterwards, confirm the installation by typing "`mcut --help`" at the command prompt, if the help message of mcut command is shown, installation is completed.

```
$ git clone http://scm.sourceforge.jp/gitroot/nysol/mcmd.git
$ cd mcmd
$ ./configure
$ make
$ sudo make install
```

Chapter 2

Class

2.1 Mcsvin - Read CSV Class

This class process CSV data file by row. The features are as follows.

- Implemented in C++ and thus operate at high speed.
- If the first row of data consists of field names, field names can be stored as key in hash data.
- Hash / Array can be used for the storage of data (Array is 2 times faster).
- Key break processing can be handled easily.
- Loosely follow RFC 4180
- Assumed that the number of items in each row is fixed.

2.1.1 Method

* MCMD::Mcsvin::new(arguments){block}

Create Mcsvin object. Specify the list of arguments delimited by space in character string at " `arguments` " as follows.

`i=`	Input file name (String) [required]
`k=`	Detect key break in the list of fields. Multiple fields are delimited by comma. Note the specification of key depends on yield arguments from each method.
`-nfn`	No field names in the first row.
`-array`	Store each data field by each method in Array. Data fields are stored in Hash by default if this is not specified. Storage in Array is 2 times more efficient than Hash (refer to benchmark for details).
`block`	Execute (yield) when block is specified.

* MCMD::Mcsvin::each{|val| block}
* MCMD::Mcsvin::each{|val,top,bot| block}

Process CSV file one row at a time. 1) val is set to the value when key (k=) is not specified. 2) when key is specified, key break information is set in top and bot variables with the exception of val.

val Set value in field name as key in Hash (or Array). Values are stored in string format.

top Set to true if the start of key is specified at k=, otherwise, set to false. See remarks for more details.

bot Set to true if the end of key is specified at k=, otherwise, set to false. See remarks for more details.

* MCMD::Mcsvin::names()

Return field name array. Return nil if `-nfn` is specified.

2.1.2 Remarks

- Store data in Array if `-nfn` is specified. Note that data cannot be stored in Hash.
- The specified field defined at `k=` must be sorted beforehand.
- Logic of key break:

```
MCMD::Mcsvin.new("i=input.dat k=key"){|csv|
csv.each{|val,top,bot|
    :
  }
}
```

In the above code, the logic of top and bot settings in bool type block variable is as follows.
Data row $i = 1, 2, \cdots, n$, value of key field ("key") in row i where k_i can simply be expressed as $k_0 = k_{n+1} = \phi$. Given $k_i (1 \leq i \leq n) \neq \phi$.

$$\texttt{top} = \begin{cases} true, & \text{if } k_i \neq k_{i-1} \\ false, & \text{otherwise} \end{cases} \tag{2.1}$$

$$\texttt{bot} = \begin{cases} true, & \text{if } k_i \neq k_{i+1} \\ false, & \text{otherwise} \end{cases} \tag{2.2}$$

2.1.3 Example

Example 1 Return the row number and values for corresponding field names

```
# dat1.csv
customer,date,amount
A,20081201,10
B,20081002,40

MCMD::Mcsvin.new("i=dat1.csv"){|csv|
  puts csv.names.join(",")
  csv.each{|val|
    p val
  }
}
# Output results
customer,date,amount
["customer"=>"A", "date"=>"20081201", "amount"=>"10"]
["customer"=>"B", "date"=>"20081002", "amount"=>"40"]
```

Example 2 Key break processing

```
# dat1.csv
customer,date
A,20081201
A,20081202
B,20081003
C,20081004
C,20081005
C,20081006

csv=MCMD::Mcsvin.new("i=dat1.csv k=customer")
csv.each{|val,top,bot|
  puts "#{val['customer']},#{val['date']} top=#{top} bot=#{bot}"
}
csv.close

# Output results
A,20081201 top=true bot=false
A,20081202 top=false bot=true
B,20081003 top=true bot=true
C,20081004 top=true bot=false
C,20081005 top=false bot=false
C,20081006 top=false bot=true
```

Example 3 Data processing without field names

Data is stored in Array when `-nfn` is specified.

```
# dat1.csv
A,20081201
A,20081202

MCMD::Mcsvin.new("i=dat1.csv k=1 -nfn"){|csv|
```

```
  puts csv.names # -> nil
  csv.each{|val|
    p val
  }
}

# Output results
nil
["A","20081201"]
["A","20081202"]
```

Example 4 Example of storing data in Array

```
# dat1.csv
customer,date,amount
A,20081201,10
B,20081002,40

# Array storage with -array option
MCMD::Mcsvin.new("i=dat1.csv -array"){|csv|
  puts csv.names.join(",")
  csv.each{|val|
    p val
  }
}

# Output results
customer,date,amount
["A", "20081201", "10"]
["B", "20081002", "40"]
```

Related Commands

Mcsvout : Write to CSV data.

Mtable : Read data by cell from CSV file.

2.1.4 Benchmark Test

The processing speed for various Ruby extension library are benchmarked in terms of reading CSV data. The benchmark test targets the following 4 libraries and mcut command.

CSVScan http://raa.ruby-lang.org/project/csvscan/

LightCsv http://tmtm.org/ruby/lightcsv/

FasterCSV http://www.gesource.jp/programming/ruby/database/fastercsv.html

CSV http://www.ruby-lang.org/ja/old-man/html/CSV.html

mcut Fields are extracted with M-Command (implemented in C++). Speed performance is shown as reference.

The results of benchmark test is shown in Table 2.1. In this experiment, 10,000 to 500 million rows of data is read. The performance of Mscvin is almost equivalent to CSVScan (implemented in C). The difference is more significant for other libraries implemented in Ruby native code. However, Mcsvin lags behind when compared with mcut. The same parsing logic for CSV is used for mcut and Mcsvin, the difference due to the cost incurred when data is store in Array in Ruby interface. An excerpt of the script used in the benchmark test is shown in Figure 2.1.

The next test looks at the difference in execution time corresponding to the types of data storage for data with or without key (Table 2.2). There is minimal difference in speed for data with or without key, however, the speed of data stored in Array is twice as fast as Hash.

Table 2.1: Comparison of execution speed among various CSV libraries (in seconds)

Number of rows	10K	100K	1000K	2000K	3000K	4000K	5000K
Mcsvin	0.020	0.196	1.76	3.51	5.26	7.02	8.79
CSVScan	0.021	0.187	1.83	3.67	5.50	7.33	9.17
LightCsv	0.155	1.62	15.99	–	–	–	–
FasterCSV	0.196	1.96	19.50	–	–	–	–
CSV	1.44	14.3	–	–	–	–	–
mcut	–	–	0.095	0.177	0.260	0.342	0.423

The results show the average value (real time) of 10 benchmark tests.
Benchmark is not measured in cells marked as " - " since the value is too big (or too small).
10K number of rows refer to 10,000 rows. Data size of 1000K records is about 25MB. The data consists of 5 columns.
Version: CSVScan 0.0.20070920, FasterCSV 1.5.1, LightCsv 0.2.2 CSV(Ruby 1.8.7)
Test environment: Mac Book Pro, 2.66GHz Intel Core i7, 8GB memory, Mac OS X 10.6.8

```
require 'rubygems'
require 'csv'
require 'fastercsv'
require 'lightcsv'
require 'csvscan'
require 'mcmd'

require 'benchmark'

puts Benchmark.measure{
  (0...10).each{|i|
    # Case of Mcsvin
    csv=MCMD::Mcsvin.new("i=data.csv -array"){|csv| csv.each{|val|}}

    # Case of CSVScan
    File.open("data.csv","r"){|fp| CSVScan.scan(fp){|row|}}

    # Case of LightCsv
    LightCsv.foreach("data.csv"){|row|}

    # Case of FasterCSV
    FasterCSV.foreach("data.csv"){|row|}

    # Case of CSV
    CSV.open("data.csv", 'r'){|row|}
  }
}
```

Figure 2.1: Excerpt of benchmark test script

Table 2.2: Comparison of execution speed according to data types with and without key (in seconds)

Key	Type	1000K	2000K	3000K	4000K	5000K
No	Array	1.76	3.51	5.26	7.02	8.79
No	Hash	3.52	6.99	10.50	14.00	17.52
Yes	Array	1.97	3.92	5.88	7.84	9.83
Yes	Hash	3.68	7.34	11.01	14.73	18.34

Size of key at an average of 10 rows.

2.2 Mcsvout - Write CSV Class

This class returns CSV data file with the following features.

- Implemented in C++ and thus operate at high speed.
- Handle any format with or without field names.
- Loosely follow RFC 4180.
- Assumed that the number of items in each row is fixed.

2.2.1 Method

* MCMD::Mcsvout::new(arguments){block}

Create Mcsvout object. Specify the following arguments as character string delimited by space at `arguments`.

`o=`	Output file name (String)
`f=`	Set the array of field names in character string as header (first line) of CSV output data. When `size=` is specified without `f=`, the CSV output will not include field names.
`size=`	Specify the number (Fixnum) of columns in CSV when field names is not present in output.
`precision=`	Specify the number of significant digits as floating variable. Default value is 10 digits. Value in C language output format `"%.ng"`の `n` の値。 Round 100/3 to significant digits of 5 becomes 33.333, and significant digits of 2 becomes 33.
`bool=`	Specify the true and false output value separated by comma. Default value is "1,0".

* MCMD::Mcsvout::write(values)

values Return CSV data stored in array. Data classes that can be stored in the array includes String, Fixnum, Bignum, Float, nil, true, false. All other classes will be treated as nil. If the size of array is less than the number of field names, null value is added to the output. If the size of array is greater than the number of field names, the excess will not be included in the output.

2.2.2 Remarks

- If comma is included in character string, the value is automatically enclosed in double quotes. Double quotes in a character string is replaced by two double quote characters.

2.2.3 Example

Example 1 Example of including row of field names in CSV output data

```
csv=MCMD::Mcsvout.new("i=rsl.csv f=a,b,c"){|csv|
  csv.write(["1",2,3.4])
  csv.write([1,2,3,4,5])
  csv.write([1,2])
}

# Output results (rsl.csv)
a,b,c
1,2,3.4
1,2,3
1,2,
```

Example 2 Example of excluding row of fields names in CSV output data

```
csv=MCMD::Mcsvout.new("i=rsl.csv size=3"){|csv|
  csv.write(["1",2,3.4])
  csv.write([true,nil,false])
```

```
  csv.write(["4\"5","","6,7"])
}

# Output results (rsl.csv)
1,2,3.4
1,,0
"4""5",,"6,7"
```

Example 3 Specify option (significant digit,bool value)

```
MCMD::Mcsvout.new("i=rsl.csv size=3 precision=3 bools=T,F"){|csv|
  csv.write([0.123456,123456.0]) # Note that the decimals beyond the specified significant digits are not
  csv.write([123456,0])  # Specifying the number of significant digits does not affect  Fixnum
  csv.write([true,false])
}

# Output results (rsl.csv)
0.123,1.23e+05
123456,0
T,F
```

Example 4 Copy data

```
# dat1.csv
customer,date
A,20081201
B,20081002

MCMD::Mcsvin.new("i=dat1.csv -array"){|csvIn|
  MCMD::Mcsvout.new("i=rsl.csv f=#{csvIn.names.join(",")}){|csvOut|
    csvIn.each{|val|
      csvOut.write(val)
    }
  }
}

# rsl.csv
customer,date
A,20081201
B,20081002
```

2.2.4 Benchmark Test

The processing speed for various Ruby extension library are benchmarked in terms of writing CSV data. Two libraries are benchmarked as follows.

FasterCSV http://www.gesource.jp/programming/ruby/database/fastercsv.html

CSV http://www.ruby-lang.org/ja/old-man/html/CSV.html

The results of benchmark test is shown in Table 2.3. 1 million rows, 10 million rows, and 100 million rows of data is written for the experiment. However, the data is not written to an actual file, instead, it is printed to null device (/dev/null). An excerpt of the benchmark test script is shown in Figure 2.2. Since Mcsvout is implemented in C++, its processing speed is faster than the other two libraries. The difference is because the other two libraries are implemented in Ruby native code.

2.2.5 Related Command

Mcsvin : Read from CSV data.

Table 2.3: Comparison of execution speed among various CSV libraries (in seconds)

Number of rows	10K	100K	1000K
Mcsvout	0.0158	0.150	1.50
FasterCSV	0.232	1.90	20.0
CSV	0.279	2.80	27.9

The results show the average value (real time) of 10 benchmark tests.
10K number of rows refers to 10,000 rows. 6 types of values including String, Fixnum, Float, true, false, nil are returned in output.
Version: FasterCSV 1.5.1 CSV(Ruby 1.8.7)
Test environment: Mac Book Pro, 2.66GHz Intel Core i7, 8GB memory, Mac OS X 10.6.8

```
require 'rubygems'
require 'csv'
require 'fastercsv'
require 'mtools'

require 'benchmark'

$data = ["12345678", 10, 1.1, true, nil, false]

puts Benchmark.measure{
  (0...10).each{|i|
    # Case of Mcsvout
    MCMD::Mcsvout.new("o=/dev/null size=6){|csv|
      (0...10000).each{|j|
        csv.write($data)
      }
    }

    # Case of FasterCSV
    FasterCSV.open("/dev/null", 'w'){|csv|
      (0...10000).each{|j|
        csv << $data
      }
    }

    # Case of CSV
    CSV.open("/dev/null", 'w'){|csv|
      (0...10000).each{|j|
        csv << $data
      }
    }
  }
}
```

Figure 2.2: Excerpt of benchmark test script

2.3 Margs - Argument operation class

This class handles arguments at command line. The features are as follows.

- Handle 2 types of argument format namely `keyword=value` and `-keyword`.
- Convert Bool type (true/false) data using `-keyword` option.
- Defined format of `value` could be in Ruby original String array, Fixnum array, Float array.
- Provide file type and field name type for other special format.
- Specify default value and value range.
- Display error message if the specified argument is not correct.
- When `--help` is specified, help() is called. User can also specify the function name of help. Note that `--help` is different than regular help option, and the prefix includes two minus sign.

2.3.1 Method

* MCMD::Margs.new(ARGV[,allKeyWords][,mandatoryKeyWords][,helpFunction)

Create Margs object. The arguments at the command line expressed in " `keyword=value` " or " `-keyword` " are set to Array or Hash variable within the class.

`ARGV` - ARGV variable of Ruby.

`allKeyWords` - Define argument keyword list by `key=value` or `-key`. Check the arguments other than the ones defined here is not defined at ARGV, if the argument is specified, the program will terminate with an error. Checking will not be carried out when `allKeyWords` is not defined.

`mandatoryKeyWords` - Define argument keyword list by `key=value`. If the specified arguments is not specified at the command line, the program terminates with an error. Checking will not be carried out when `mandatoryKeyWords` is not defined.

`helpFunction` - Function name to call when `--help` is specified.

```
# Command line
$ ruby test.rb i=dat.csv -abc

# Contents of test.rb
args=Margs.new(ARGV, "i=,v=,-abc") # OK
args=Margs.new(ARGV, "i=,v=") # -abc cannot be specified and it is terminated with error
args=Margs.new(ARGV, "i=,v=,-abc","i=,v=") # v= is required but is not specified,
thus it is terminated with error
```

* MCMD::Margs.file(keyWord,mode): Obtain file name

`keyWord` - Keyword in `key=`format (String)

`mode` - Specify "r"(check readable) or "w"(check writable)(String)

The value specified at keyWord is the input filename, when mode is set as "r", the method returns the file name if the file is readable. Otherwise, if the file cannot be read, it terminates with an error. When mode is set as "w", check whether the directory is writable, and return filename if it is writable, otherwise, terminate with error if not writable.

```
# Command line
$ ruby test.rb i=dat.csv

# Contents of test.rb
args=Margs.new(ARGV)
puts args.file("i=","r") # dat.csv becomes "dat.csv" if it is readable
puts args.file("i=","w") # current directory is "dat.csv" if it is writable
```

* MCMD::Margs.field(keyWord,fileName)

Return various information in the file (specified at `fileName`) related to the field names specified at `keyWord`.

`keyWord` - Keyword in `"key="` format (String).

`fileName` - File name.

The specified field name at the command line must follow the format below.

$$\texttt{key=}name_1\texttt{[:}newName_1\texttt{\%}option_1\texttt{]},name_2\texttt{[:}newName_2\texttt{\%}option_2\texttt{]},\ldots \tag{2.3}$$

Specify multiple field names delimited by comma. $name_i$ represents the field names in the CSV file which is specified at `fileName`. Otherwise, terminate with the error `"field name not found"`.

The two attributes $newName_i$ and $option_i$ can be specified (optional) at field name $name_i$. The attributes have various uses, and must be separated by `:` and `%`.

Typically, the calculation results from fields $name_i$ will be added as a new field name in the output $newName_i$. In addition, $option_i$ is an option used to control the content of processing.

This method returns a variety of information shown below in Hash. Font in Bold represents Hash key.

`names` - `names` is the field name of the array (String Array).

`newNames` - `newNames` is the new field name of the array (String Array). nil if this is not specified.

`options` - `options` is the option of the array (String Array). nil if this is not specified.

`fld2csv` - Item number (start from 0) in the CSV file (`fileName`) which corresponds to the fields specified at `"key="` (String Array).

`csv2fld` - Field number in CSV file is numbered according to the fields specified at `"key="` in sequential order (starting from 0) (String Array). Fields that are not specified is nil.

```
# Contents of test.csv
fld1,fld2,fld3
1,2,3
4,5,6

# Command line
$ ruby test.rb f=fld1,fld3

# Contents of test.rb
args=Margs.new(ARGV)
fld=args.field("f=","test.csv")
p fld["names"]   # -> ["fld1","fld3"]
p fld["fld2csv"] # -> [0,2] fld1,fld3 corresponds to 0th item and 2nd item in test.csv
p fld["csv2fld"] # -> [0,nil,1] the 0th item in test.csv is specified as 0th item at f=

# Command line
$ ruby test.rb f=fld3:newFld3%n,fld2%nr

# Contents of test.rb
args=Margs.new(ARGV)
fld=args.field("f=","test.csv")
p fld["names"]    # -> ["fld3", "fld2"]
p fld["newNames"] # -> ["newFld3", nil]
p fld["options"]  # -> ["n", "nr"]
p fld["fld2csv"]  # -> [2, 1]
p fld["csv2fld"]  # -> [nil, 1, 0]
```

* MCMD::Margs.str(keyWord[,default][,token1][,token2])

Obtain character string arguments

`keyWord` - Keyword in `"key="` format (String)

`default` - Default value when the value is not specified. The value is nil if not specified.

`token1` Delimiter when multiple character strings are specified. When the argument is not specified, no delimiter will be used.

`token2` The character string divided by token1 is further delimited by token2. There will not be delimiter if the argument is not specified.

Among the arguments specified at the command line, return the character string that matches `keyWord`.

Return the `default` character string specified at the command line. Returns nil if `default` contains `nil`.

Return array when `token1` is specified as the delimiter of the character string. If `token2` is specified, return array within array.

```
# Command line
$ ruby test.rb v=abc

# Contents of test.rb
args=Margs.new(ARGV)
puts args.str("v=") # ->"abc"
puts args.str("w=") # -> nil
puts args.str("w=","xyz") # -> "xyz"
```

```
# Command line
$ ruby test.rb v=abc,efg:xyz,hij

# Contents of test.rb
args=Margs.new(ARGV)
puts args.str("v=") # ->"abc,efg:xyz,hij"
puts args.str("v=",nil,",") # ->["abc", "efg:xyz", "hij"]
puts args.str("v=",nil,",",":") # ->[["abc"], ["efg","xyz"], ["hij"]]
```

* MCMD::Margs.float(keyWord[,default][,from][,to]): Obtain float type numerical arguments

`keyWord` - Keyword in "key=" format (String).

`default` - Specify the default value (Float). The value is nil if not specified.

`from` Lower limit (Float) of range check. Lower limit is not checked if the argument is not specified.

`to` Upper limit (Float) of range check. Upper limit is not checked if the argument is not specified.

For all arguments specified at command line, convert values that matches `keyWord` to Float. Return the `default` value specified at the command line. Program will terminate with an error if the range check fails.

```
# Command line
$ ruby test.rb v=0.12

# Contents of test.rb
args=Margs.new(ARGV)
puts args.float("v=") # -> 0.12
puts args.float("v=",nil,0.2,0.3) # -> Range error
puts args.float("w=") # -> nil
puts args.float("w=",0.1) # -> 0.1
```

* MCMD::Margs.int(keyWord[,default][,from][,to]) Obtain fixnum type numerical arguments

`keyWord` - Keyword in `key=` format (String).

`default` - Specify the default value (Float). The value is nil if not specified.

`from` - Lower limit (Float) range check. Lower limit is not checked if the argument is not specified.

`to` - Upper limit (Float) range check. Upper limit is not checked if the argument is not specified.

Among all arguments specified at command line, for values that matches `keyWord`, convert to Float. Return the `default` value specified at the command line. Program terminates with an error if range check fails.

```
# Command line
$ ruby test.rb v=10

# Contents of test.rb
args=Margs.new(ARGV)
puts args.int("v=") # -> 10
puts args.int("v=",,20,30) # -> Range error
```

```
puts args.int("w=") # -> nil
puts args.int("w=",15) # -> 15
```

* MCMD::Margs.bool(keyWord) Obtain bool type arguments

`keyWord` Keyword that corresponds to "-key" (String)

Among the arguments specified at the command line, return true if `keyWord` matches with the argument, otherwise return false.

```
# Command line
$ ruby test.rb -flag

# Contents of test.rb
args=Margs.new(ARGV)
puts args.bool("-flag") # -> true
puts args.bool("-x") # -> false
```

2.3.2 Example

Example 1

```
# Command line
$ ruby test.rb i=dat.csv v=value -abc

# Contents of test.rb
args=Margs.new(ARGV, "i=,o=,w=,-flag,-x", "i=,w=")
iFileName = args.file("i=") # -> "dat.csv"
oFileName = args.str("o=","result.csv") # -> "result.csv"
weight    = args.float("w=",0.1,0.0,1.0) # -> 0.1
flag      = args.bool("-abc") # -> true
wFlag     = args.bool("-w") # -> false
```

2.3.3 Related Command

2.4 Mtable - Read CSV data into cell class

This class read specified CSV data into random access memory by cell. Its features are as follows.

- Access cells in random order by the specified row and column.
- Data is read-only and cannot insert or update records.
- All data is read as character string. Corresponding type conversion (ex. to_i) is necessary for other usage.
- Read data into available memory. The program terminates with an error when there is insufficient space.

2.4.1 Method

* MCMD::Mtable::new(arguments)

Generate Mtable object. At the `arguments` parameter, specify the following arguments in character string separated with a space.

`i=`	Input file name (String)
`-nfn`	No field name in the first row.

* MCMD::Mtable::cell(col=0, row=0) -> String

Return the value of cell to the corresponding `row` and `col` (column). `row` and `col` is assigned by sequential row and column number. Both row and column number are expressed in integer starting from 0 (Column number of Mcsvin starts from 1). Returns nil if `row` and `col` are out of range.

row Row number. Positive integer value including 0. Default value is 0.

col Column number. Positive integer value including 0. Field name cannot be specified. Default value is 0.

Returns the value of col number of the column as `col`. It is equivalent to specifying cell(col, 0). However, if both `col` and `row` is not specified, it returns 0th item of 0th row. It is equivalent to specifying cell(0,0).

* MCMD::Mtable::names() -> String Array

Return the field name array.

* MCMD::Mtable::name2num() -> String=>Fixnum Hash

Return Hash of key of field name and the the corresponding field number.

* MCMD::Mtable::size() -> Fixnum

Return the number of rows.

2.4.2 Examples

Example 1

```
# dat1.csv
customer,date,amount
A,20081201,10
B,20081002,40

tbl=MCMD::Mtable.new("i=dat1.csv")
p tbl.names     # -> ["customer", "date", "amount"]
p tbl.name2num  # -> {"amount"=>2, "date"=>1, "customer"=>0}
p tbl.size      # -> 2
p tbl.cell(0,0) # -> "A"
p tbl.cell(0,1) # -> "B"
p tbl.cell(1,1) # -> "20081202"
```

```
p tbl.cell(1)   # -> "20081201"
p tbl.cell      # -> "A"
```

Example 2 Without column names in the first row

```
# dat1.csv
customer,date,amount
A,20081201,10
B,20081002,40

tbl=MCMD::Mtable.new("i=dat1.csv -nfn") # Treats the first row as data.
p tbl.names     # -> nil
p tbl.name2num  # -> nil
p tbl.size      # -> 3
p tbl.cell(0,0) # -> "customer"
p tbl.cell(0,1) # -> "A"
p tbl.cell(1,1) # -> "20081201"
p tbl.cell(1)   # -> "date"
p tbl.cell      # -> "customer"
```

2.4.3 Related Command

Mcsvin : Read CSV data.

Chapter 3

Module Methods

3.1 meach - Execution method for simple parallel processing

Method of Array class for simple parallel processing without exclusive control. It is a simple implementation to initiate multiple processes asynchronously.

3.1.1 Format

```
1) Array::meach(mpcount){|value| block}
2) Array::meach(mpcount){|value,count| block}
```

Process the code in parallel by `block` when `value` is given as the block parameter of an element in an array. If `count` is given as a block parameter, set the array element number (integer from 0) being processed.

`mpcount` - Number of parallel processes.

3.1.2 Examples

Example 1 Print row number and value from field name

```
# Process five elements from 10 to 6 (integer) in two parallel threads.
# Processing content is as simple as displaying the value of element and the element
number of the array.
> [10,9,8,7,6].meach(2){|value,count|
>   puts "#{value},#{count}"
> }
10,0
9,1
8,2
7,3
6,4
```

3.1.3 Related Command

3.2 mheader - Acqusition method for field name array in CSV data

Return the field names (in the first row) of the CSV data file in an array. For files without field names in the first row, return the value of each field in the first row in an array.

3.2.1 Format

`MCMD::mheader(arguments)`

Specify the following arguments in character string separated with space at `arguments`.

`i=` Input file name (String)

3.2.2 Example

Example 1

```
# dat1.csv
# customer,date,amount
# A,20081201,10
# B,20081002,40

> p MCMD::mheader("i=dat1.csv")
 => ["customer", "date", "amount"]
```

Example 2

```
# dat1.csv
# A,20081201,10
# B,20081002,40

> p MCMD::mheader("i=dat1.csv")
 => ["A", "20081201", "10"]
```

3.2.3 Related Commands

Mcsvin : Class for reading CSV data.

Mtable : Class for reading CSV data into cell.

3.3 mrecount - Row calculation method for CSV data

Class to process CSV data file by row. The features are as follows.

- Implemented in C++ and thus operates at high speed (Slightly faster than the UNIX command `wc-l`).
- Only counts the number of rows in data excluding the field names in the first row.
- Only counts the line break char, thus, line break(s) escaped by double quotes are also counted. Use MCMD::Mtable to avoid this problem.

3.3.1 Format

`MCMD::mrecount(arguments)`

Specify the following arguments in character string separated with space at `arguments`.

`i=`	Input file name (String)
`-nfn`	No field names in the first row.

3.3.2 Examples

Example 1 Print row number and value from field names

```
# dat1.csv
customer,date,amount
A,20081201,10
B,20081002,40

p MCMD::mrecount("i=dat1.csv")      # -> 2
p MCMD::mrecount("i=dat1.csv -nfn") # -> 3
```

3.3.3 Related Command

Mtable : Class to read CSV data into cell

3.3.4 Benchmark Test

The processing speed of the UNIX command `wc` and `Mtable` are benchmarked in terms of row count of CSV data. The results of the benchmark test in shown in Table 3.1. The experiment is carried out for data with one million, two million, three million, four million and 500 million rows. As shown in the results, `mrecount` is slightly faster than `wc`. Further, `Mtable` is not a class for used to count the number of rows, mrecount is 5-6 times faster when compared to Mtable.

An excerpt of the script used in the benchmark test is shown in Figure 3.1.

Table 3.1: Comparison of execution speed among various CSV library (in seconds)

Number of rows	1000K	2000K	3000K	4000K	5000K
mrecount	0.034	0.066	0.097	0.129	0.161
wc -l	0.038	0.070	0.103	0.133	0.169
Mtable	0.231	0.407	0.503	0.731	0.828

The results show the average value (real time) of 10 benchmark tests.
1000K number of rows refer to one million rows. Data size of 1000K records is about 25MB. The data consists of 5 columns.
Test environment: Mac Book Pro, 2.66GHz Intel Core i7, 8GB memory, Mac OS X 10.6.8

```
require 'rubygems'
require 'mtools'

require 'benchmark'

puts Benchmark.measure{
  (0...10).each{|i|
    # Case of mrecount
    p MCMD::mrecount("i=data.csv")

    # Case of wc
    system "wc -l data.csv"

    # Case of Mtable
    MCMD::Mtable("i=data.csv -array"){|tbl|
                        p tbl.recordSize
    }
  }
}
```

Figure 3.1: Excerpt of benchmark test script

3.4 mkDir - Create directory method

Create directory. If the directory specified at the parameter exists, it is removed and created.

3.4.1 Format

`MCMD::mkDir(dirName[,rmExistingDir])`

`dirName` - Directory name to create.

`rmExistingDir` - When true is specified, if `dirName` exists, it is removed and created.

3.4.2 Examples

Example 1 Basic Example

```
# Create the directory ./folder.
# Do not do anything if the directory exists.
> MCMD::mkDir("./folder")
# When true is specified at the second,  if ./folder already exists, delete the existing
 directory and create a directory .
> MCMD::mkDir("./folder",true)
```

3.4.3 Related Command

3.5 endLog - Output log message upon completion

Display log message which is similar to the format of regular output log messages of MCMD. The format is as follows.

```
#END# message; date and time
```

3.5.1 Format

* `MCMD::endLog(msg[,fileObject])`

  `msg` - Message to be displayed.

  `fileObject` - Output file object. Returns standard error (STDERR) when this is not specified.

3.5.2 Examples

Example 1 Basic Example

```
# Display ending error message to standard error.
> MCMD::endLog("mburst.rb ended normally")
#END# mburst.rb  ended normally.; 2013/11/01 19:09:50
```

3.5.3 Related Commands

errorLog : Output error log message.

messageLog : Output general log message.

3.6 errorLog - Output error log message

Display error log message which is similar to the format of error log messages of MCMD shown upon completion. The format is as follows.

```
#ERROR# mesage; date and time
```

3.6.1 Format

`MCMD::errorLog(msg[,fileObject])`

`msg` - Message to be displayed.

`fileObject` - Output file object. Returns standard error (STDERR) when this is not specified.

3.6.2 Example

Example 1 Basic Example

```
# Display ending error message to standard error.
> MCMD::errorLog("mburst.rb ended with error. ")
#END# mburst.rb  ended with error.; 2013/11/01 19:09:50
```

3.6.3 Related Commands

endLog : Output end log message.

messageLog : Output general log message.

3.7 messageLog - Output end log message

Display log message which is similar to the format of normal log messages of MCMD shown upon completion. The format is as follows.

```
#MSG# message; date and time
```

3.7.1 Format

* `MCMD::messageLog(msg[,fileObject])`

  `msg` - Message to be displayed.

  `fileObject` - Output file object. Returns standard error (STDERR) when this is not specified.

3.7.2 Examples

Example 1 Basic Example

```
# Display normal message to standard error.
> MCMD::messageLog("mburst.rb This is a message. ")
#END# mburst.rb This is a message. ; 2013/11/01 19:09:50
```

3.7.3 Related Commands

errorLog : Output error log message

endLog : Output end log message

Chapter 4

Others

4.1 MCMD Module Configuration

MCMD module is built using KGMOD library. All shell variables set in KGMOD can be used. The following explains the configuration method of Ruby MCMD module.

4.1.1 Output message

The environment variable `KG_VerboseLevel` can be set to control the message to standard error for various methods such as `Mcsvin`. The parameter and details are as follows.

Table 4.1: Parameters of environment variables that control the message and its contents

Parameter	Description
0	Do not output any message
1	+ error message output
2	+ warning message output
3	+ end message output
4	+ msg message output (default)

```
$ irb
> require 'mcmd'

# By default KG_Verbose=4, both normal message and error message are displayed.
> MCMD::Mcsvin.new("i=dat.csv"){|csv| csv.each{|flds|}}
#END# mcsvin i=dat.csv; ; 2013/08/08 15:18:52
> MCMD::Mcsvin.new("x=dat.csv"){|csv| csv.each{|flds|}}
#ERROR# unknown parameter x= (mcsvin); mcsvin x=dat.csv; ; 2013/08/08 15:18:52

# When KG_Verbose=1, ending error message is displayed, but the normal ending message is not displayed.
> ENV["KG_VerboseLevel"] = "1"
> MCMD::Mcsvin.new("i=dat.csv"){|csv| csv.each{|flds|}}
> MCMD::Mcsvin.new("x=dat.csv"){|csv| csv.each{|flds|}}
#ERROR# unknown parameter x= (mcsvin); mcsvin x=dat.csv; ; 2013/08/08 15:18:52

# When KG_Verbose=0, both messages are not displayed.
> ENV["KG_VerboseLevel"] = "0"
> MCMD::Mcsvin.new("i=dat.csv"){|csv| csv.each{|flds|}}
> MCMD::Mcsvin.new("x=dat.csv"){|csv| csv.each{|flds|}}
```