Awesome Financial Software

Jonathan Kaufman, PhD

Introduction

Detailed historical company information has been disseminated via the SEC Edgar system for many years now, offering investors a convenient way to get copies of historical SEC filings related to any publicly listed company. However, there is a limitation to what can be easily found in the Edgar system, since there is often little way to know what information is contained in any given filing without first opening it. A common practice people who use the sec website to obtain company information is to start with a different tool to find out what dates certain events (such as good or bad news regarding the company's business, or issuance of additional securities to raise money) occur before referring to the sec website to search for filings. Know the date (or approximate date) of a filing that may be interesting helps a lot. Mug provides ways for identifying these dates, and then will open the relevant sec webpage (conveniently without having to search for it).

A vast amount of information used by equity traders and institutional investors is gradually becoming free (or almost free). It was not too long ago that to get consolidated high-quality investment information, an institutional investor had to purchase data access. The most well know vendors are Bloomberg and FactSet. These terminals, which provide extensive market data require annual subscriptions for thousands of dollars per month per terminal. The business models of these companies may need to shift a bit, because basic advances in technology now provide essentially the same information for free – if you can interact with the provider's application programming interface (API). Yes, the data is now free, but you have to be some kind of computer wiz to be able to obtain it and/or read it. The data often has to be decoded from a JSON format for it to be conveniently human readable. Mug provides all this data (from multiple providers: including iexcloud, alpaca trading, alpha-vantage, …) via a simple yet smart command line interface (CLI) that allows you to find patterns or outliers.

Much of financial information cannot exist in a vacuum, and thus the value or meaning of some of the data mainly relevant with respect to other data and/or conditions. For this reason, some of the most interesting financial information is in the form of computational results of presently available data. For example, occasionally an investor may want to not only know what the cash position is of a company, but additionally want to know how that value of cash compares to other companies (or rather collections of companies that have one or more aspects in common. Or the investor may

simply want to measure the cash in relation to the company's mark capitalization. Mug simplifies the process of making these comparisons by creating sets of companies to serve as comparators, as well as by providing the ability to look at ratios (and other combinations) of scalar numerical data within a company and across companies.

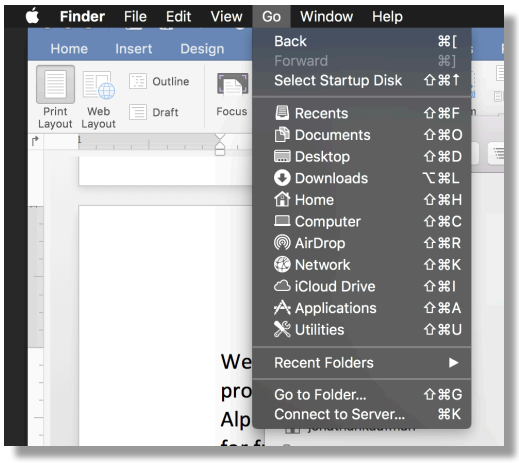
Sophisticated financial analysis is not included in the systems used by typical financial data providers. There is little if any ability to run calculations on sets of companies, such as various clustering algorithms on a set of companies. Mug provides many group analytical techniques (GATs), and is constantly expanding with new ones. Mug also provides capability of evaluating pairs of securities, which is essential for developing common statistical arbitrage trading strategies.

So give Mug a try and see if it helps you find interesting investment strategies and/or opportunities. Read further to learn some of the cool details of what you can do with Mug.

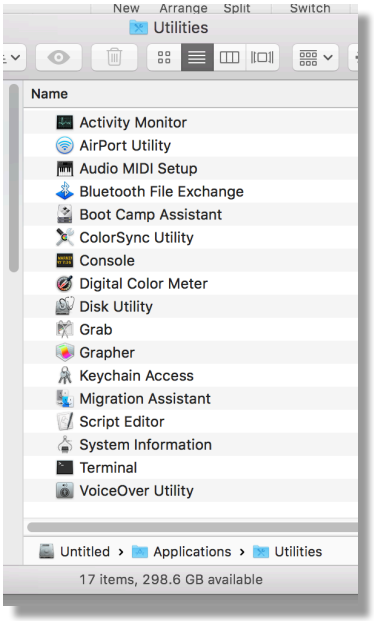
Getting Started on the Mac

Welcome to Mug. Awesome financial software for exploring equities. Mug currently provides smart interfaces to several financial data providers, including IEX Cloud, Alpaca, and Alphavantage.

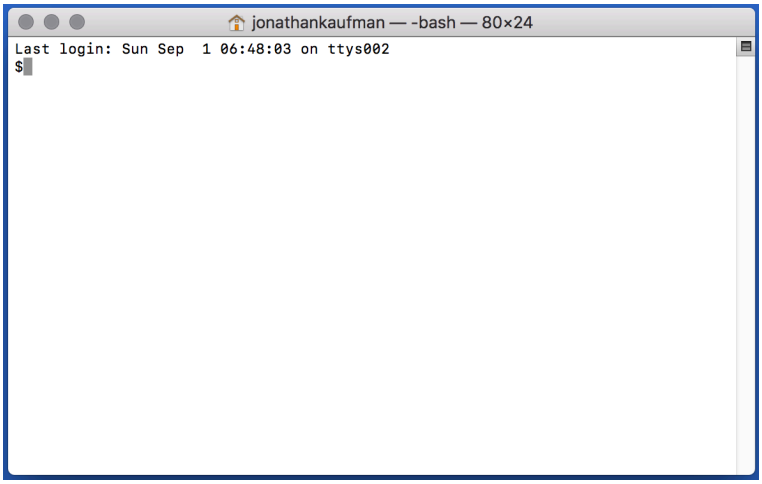
Mug is currently provided as a command line application. Command line applications are accessed via a terminal window. To access a terminal window from a mac os x system, you can use the "Go" dropdown menu when having the system Finder in focus.



Then choose the "Utilities" menu item in the drop-down list. Note the utilities can also be opened with shift-command-U. When doing so, you should get a folder with similar contents as shown in figure TK.



Double-click the Terminal icon to open a terminal window. This terminal window will provide the command-line for the Mug command line interface (CLI).



First, however, we need to navigate our terminal window to the folder where the Mug application is located. This can be accomplished with the unix cd (cd for change directory) command. On my computer, the location is

```
$ cd /Users/jonathankaufman/Dropbox/Mug
```

Note that the "**$**" is the unix prompt at the command line window. This prompt will vary depending on your computer's settings. You should navigate to where you put the folder containing the Mug application as well as its associated data files. Then, once your terminal command line displays the correct folder location (which can be checked with the **pwd** (print working directory) command), you can (add your credentials such as secret keys for the services you will be using: iexcloud, alpaca, etc., and) initiate the Mug. Application with the following commend.

```
$ java –jar Mug.jar
```

The use of the java command indicates that Mug currently runs on the Java Virtual Machine (JVM). If Java is not installed on your computer, the java command will of course not work. To install the latest version of Java on your computer, see the detailed instructions located at java.com.

To add your iexcloud credentials to Mug, open the file named keychain.clj in the resources subdirectory. The contents of the file look like as follows.

```
{:iexcloud         "pk_c38629bfd8mdfbddc62f3356"
 :alpaca1          "PKNAEQDJFSH6UXU1O0H"
 :alpaca2          "wx4n60KMWJYb0idmnvxdcmnHtf"
 :alphavantage     "8T1KDJSDNVDYCWK"}
```

It is readily apparent where to put your secret keys and other credentials so that they can be used by Mug. The Mug software will use this file to access your accounts on the listed financial data providers.

Once Java and Mug are correctly installed, and your credentials entered into the keychain.clj file, the following prompt should appear when you first launch the Mug program using the java command indicated above (`$ java –jar mug.jar`).

```
Welcome to Mug!
***************

(For help, enter .h (including the dot) at the prompt >)

top>
```

The "`top>`" character is the top-level prompt where you provide instructions to Mug. Most commands at this prompt begin with the period symbol "**.**" For example, if you want to see a list of companies within a specific market cap range (or window), use the window command which is simply **.w** followed by the low and high limits of the range (in millions USD). So, for example, the command

```
top> .w 100 200
```

provides a list of companies within the market cap range from $100 million to $200 million. When doing so, the Mug system change state. This is evident by the new prompt which looks like

```
:fresh>
```

The colon indicates that we are no longer in the top-level state of Mug, but are alternatively in a collection. The word **fresh** in the prompt indicates that the name of the collection that is loaded is "`fresh`." This is the default name. You can create a name for any collection that is in focus using the **.s** (save) command.

```
:fresh> .s green
:green>
```

Here we see that the **.s green** command named the collection "green," and the temporary name, "fresh," remains available for the generation of new collection. Use **.l** (l for list) to get a list items in a collection (the same **.l** command provides a list of collection names when used at the top level), and use **.h** (h for help) to get a list of additional commands.

If a command is not proceeded by a "**.**" Mug interprets the text to be either a symbol name or the name of a named collection – and brings such symbol or collection into focus. Assuming we typed in the symbol mrk (case insensitive) at the prompt, the prompt updates and now starts with a "**.**" Rather than a "**:**". The "**.**" At the beginning of the prompt indicates that the Mug focus is now on a single company rather than a collection of companies (which would be indicated by the prompt starting with a "**:**". And also the symbol for this single company is displayed in the prompt so that the Mug user can be aware of state.

```
top>    .w 100 200
:fresh> .s green
:green> mrk
.MRK>
```

When Mug has a single company in focus, additional "**.**"-preceded commands are available that provide data on the company in focus. For example, to find the cash position, use **.c**, and for the market cap, use **.mkt**. The complete list of such commands is shown when typing **.h** at the prompt when a single company is in focus (which as mentioned is indicated by the prompt beginning with a "**.**" rather than "**:**").

Making Tables

Collections can be transformed into tables using the **.m** (m for map) command when a collection is in focus. Let's say we have a collection named fresh and it is in focus. (A simple way to obtain such state is to give the **.w**

(w for window) command at the top level). You can obtain a table of say each company's market cap and cash position as follows.

```
:fresh> .m c mkt
```

```
t  c     mkt
CRNX     163   634
AKBA     321   714
LXRX     160   747
KURA     178   590
SRNE     168   614
CTST     52    556
VKTX     301   600
```

Here we have "mapped" (using **.m** command) the functions **c** for cash and **mkt** for market cap. Notice that unlike the **.m** command, the **c** and **mkt** functions are not preceded with a "**.**". We can have any number of functions we like. The following table simply adds volume data.

```
:fresh> .m c mkt v
```

```
t  c     mkt   v
CRNX     163   634   0.88
AKBA     321   714   5.06
LXRX     160   747   0.89
KURA     178   590   2.47
SRNE     168   614   4.11
CTST     52    556   6.51
VKTX     301   600   26.56
```

Notice how potential outliers can be obtained with such tables, witness the volume for VKTX. We can explore if that may be related to revenue as follows.

```
:fresh> .m c mkt v r
```

```
t  c     mkt   v     r
CRNX     163   634   0.88  0
AKBA     321   714   5.06  207
LXRX     160   747   0.89  63
KURA     178   590   2.47  0
SRNE     168   614   4.11  21
CTST     52    556   6.51  35
VKTX     301   600   26.56 0
```

Here we discover no link to revenue between revenue and volume in this collection. We may be interested in mapping arithmetic combinations of these functions. For example, the cash to market cap ratio. This goal can be accomplished by putting the desired arithmetic operator (**+**, **–**, *****,or **!**) between the two functions. Notice importantly that we use the symbol **!** for division rather than a slash. Here's how simple is it to create the table.

```
:fresh> .m c mkt v c!mkt
```

```
t  c     mkt   v     c!mkt
CRNX     163   634   0.88  0.25
AKBA     321   714   5.06  0.44
LXRX     160   747   0.89  0.21
KURA     178   590   2.47  0.3
SRNE     168   614   4.11  0.27
CTST     52    556   6.51  0.09
VKTX     301   600   26.56 0.5
```

Sometimes the cash over market cap **c!mkt** values significantly exceed 1. This is usually only for microcap companies and/or companies that have completely lost the value of other assets (say with the failure of a clinical trial). Here, in this collection, the highest is 0.5 for VKTX. We can get additional information on a single security when a collection is in focus by simply including the functions after the security name when responding to the collection prompt. For example, suppose we want a list of institutional ownership for VKTX. We can use the **io** command.

```
:fresh> vktx io
```

```
2019 Vanguard_Group_Inc__Subfiler_
2019 State_Street_Corp
2019 Sio_Capital_Management_LLC
2019 Park_West_Asset_Management_LLC
2019 Northern_Trust_Corp
2019 Fiera_Capital_Corp
2019 Fidelity_Management_&_Research_Co
2019 BlackRock_Institutional_Trust_Co_NA
2019 BlackRock_Fund_Advisors
2019 ArrowMark_Colorado_Holdings_LLC
```

Or perhaps we are interested in financing cash flow (**cffd** cash flow financing detail).

```
:fresh> vktx cffd
```

```
20190630 0.26
```

```
20190331 0.32
20181231 0.34
20180930 167.48
20180630 70.14
20180331 62.29
20171231 14.76
20170930 1.11
20170630 4.0
20170331 2.52
20161231 1.03
20160930 0.1
```

Here we see a history of financing cash flow by quarter, the largest such cash flow occurring in the quarter ending September 30, 2018. Suppose you want a function that provides an indication of the extent of financing cash flow to be used in the **.m** function when creating a table. For this purpose, you can use the scalar-value **cff** instead of **cffd**, as follows.

```
:fresh> .m c mkt cff
t  c     mkt   cff
CRNX     163   634   170
AKBA     321   714   212
LXRX     160   747   147
KURA     178   590   296
SRNE     168   614   436
CTST     52    556   127
VKTX     301   600   324
```

Here, the **cff** function *reduces* the result of **cffd** so that its result can be used as a single number, which is a requirement for creating a simple table and thus a requirement for use in the **.m** command at a collection prompt. In this case, the sum of all the reported quarterly cash flows is provided. A list of such functions can be obtained at the command prompt by typing the composite help command **.h cmd**.

Exploring Symbols

Suppose you want to focus on a single company for exploration of various functions or data that cannot be succinctly represented on maps. Then by all means, as mentioned above at the end of the "Getting Started" section, simply enter the symbol for the company without including any function in the command. Let's do that for AKBA.

```
:fresh> akba
.AKBA>
```

Notice that the prompt subsequently begins with a "**.**" and not a "**:**". This is the indicator that we have a single company in focus rather than a named collection of companies. Now that we have a single company in focus, we can use all the company-focused functions directly by treating them as *commands*, and preceding them with the "**.**" symbol.

Let's try simply showing the cash position. Now be careful to use **.c** instead of just **c** . If you leave out the **.** before the **c**, Mug will think you want to look at a company that has the symbol **C**, which currently happens to be Citigroup.

```
.AKBA> .c
321
.AKBA> c
.C> .cname
Citigroup
```

As we would expect, returning to an existing named group (or the most recent version of the automatically generated fresh group) is as easy as typing the name of the group at the prompt.

```
.C> fresh
:fresh>
```

You can also return to the most recent group that was in focus with the **.u** (u for up) *command*. This *command* will also return Mug's focus to top>, if that was the focus prior to the *company* focus. Using the **.u** command when the focus is on a collection will also bring Mug's focus to top>.

Yahoo, Edgar, and Company Websites

One of Mug's design principles is to not reinvent the wheel. If there is a free service available that offers dynamic price history charts such as yahoo finance, why re-invent that? Mug will simply open a yahoo chart window for you with the **`.yahoo`** command. This command can be used at a symbol-focused prompt, or it can be used without the preceding **`.`** if it is preceded by the symbol name at the `top>` and/or `:collection>` *level* prompts.

Similarly, Mug will open the relevant www.sec.gov website page in response to the **`.sec`** command. Regarding both convenience *commands*, in practice, they are much faster and easier to use than the alternative of manually navigating a browser window/tab to the desired page. These as well as the **`.oweb`** command that opens a browser window to the desired company's website where additional basic information can be found. These commands are often the fastest way navigate to the exact websites of interest.

Updating Sets of Companies

Earlier in this introduction we created a set of companies using the **`.w`** (w for window) command. Recall that **`.w`** took two numbers as the *low* and *high* values of market cap to include in the set. Essentially **`.w`** `50 60`, generates the set of companies having market caps within the range of $50M to $60M. Let's experiment with a narrow range so the we can look at just a few companies.

```
top> .w 60 65

:fresh> .l

OVID     Ovid Therapeutics
HTGM     HTG Molecular Diagnostics
SCPH     scPharmaceuticals
GNCA     Genocea Biosciences
ALIM     Alimera Sciences
INSY     INSYS Therapeutics
EYEN     Eyenovia
AQXP     Aquinox Pharmaceuticals

:fresh>
```

suppose we want to add a company to the set. We can do this by adding the company via its ticker symbol and the **`.a`** (a of add) command as follows.

```
:fresh> .a mrk
```

```
:fresh> .a pfe

:fresh> .l

OVID     Ovid Therapeutics
HTGM     HTG Molecular Diagnostics
SCPH     scPharmaceuticals
GNCA     Genocea Biosciences
ALIM     Alimera Sciences
INSY     INSYS Therapeutics
EYEN     Eyenovia
AQXP     Aquinox Pharmaceuticals
MRK      Merck & Co.
PFE      Pfizer Inc.

:fresh>
```

We can notice apparent differences between these company's by using the **.m** (m for map) command as earlier described.

```
:fresh> .m c r e

t  c     r     e
OVID     41    0     -52
HTGM     31    21    -15
SCPH     89    0     -29
GNCA     26    0     -38
ALIM     13    46    -7
INSY     95    82    -101
EYEN     19    0     -17
AQXP     76    0     0
MRK      8864  42446 14292
PFE      18833 53647 21673

:fresh>
```

The table above is a listing of cash (**c**), revenue (**r**), and earnings (**e**). Notice how Merck (MRK) and Pfizer (PFE) are so much larger than the other companies (obtained with the **.w** function and a $65M upper limit on market cap. This is one method for finding outliers of interest in our data tables. There may be column-specific numbers that are inconsistent with those of the members of the set.

A new set can be constructed from scratch (rather than using the **.w** (w for window) command). To do so, use the **.b** (b for bag) command. The **.b** command works by simply adding the specified stock-tickers (representative of the to-be-added companies)

```
top> .b mrk pfe msft ibm ba

:fresh> .l

MRK      Merck & Co.
PFE      Pfizer Inc.
MSFT     Microsoft Corp.
IBM      International Business Machines Corp.
BA The Boeing Co.

:fresh>
```

The syntax of this command is simply **.b** followed by the tickers of interest. In addition to adding tickers to the set (as shown earlier), tickers can similarly be removed from the set with the .d (d for delete) command. Currently the **.d** command only takes a single argument, but it will soon be expanded to take any number of arguments. The following listing is an example of msft being removed from the fresh list.

```
:fresh> .l

MRK      Merck & Co.
PFE      Pfizer Inc.
MSFT     Microsoft Corp.
IBM      International Business Machines Corp.
BA The Boeing Co.

:fresh> .d msft

:fresh> .l

MRK      Merck & Co.
PFE      Pfizer Inc.
IBM      International Business Machines Corp.
BA The Boeing Co.

:fresh>
```

Notice in the listing above, MSFT was successfully removed from the list using the **.d** command.

Creating Your Universe of Companiess

If no universe of companies is selected (as a basis for set-generating commands such as the **.w** (window) command) at the top> prompt, Mug currently uses the set of all biotechnology and pharmaceutical companies.

However, this is just a default, and the universe of companies can of course be customized.

To create a new universe, use the **.nu** (nu for new universe) command, which provides you with a universe prompt which can be used for adding entire industries and/or sectors to the universe. The industries and sectors are standardized. To get a list of industries, use the **.li** command when at the `universe>` prompt. This command not only provides a list of industries, but also provides a unique abbreviation for each – so that it will not be necessary to type the entire name of an industry to select it.

Let's take the "Airlines" industry as an example. Use of the **.li** command indicates that the relevant abbreviation for this industry is `AIR`. We can use the **.ai** command (ai for add industry) to bring all of the airlines into our inventory. The **.ai** command requires that you provide the industry abbreviation following the **.ai** and befor the return. For example, what we have decribed would be conducted as follows.

```
top> .nu

universe> .ai AIR

fresh> .l

AAL   American Airlines Group
ALGT  Allegiant Travel Co.
ALK   Alaska Air Group
AVH   Avianca Holdings SA
AZUL  Azul SA
CEA   China Eastern Airlines Corp. Ltd.
CPA   Copa Holdings SA
DAL   Delta Air Lines
GOL   GOL Linhas Aéreas Inteligentes SA
HA    Hawaiian Holdings
JBLU  JetBlue Airways Corp.
LTM   LATAM Airlines Group SA
LUV   Southwest Airlines Co.
MESA  Mesa Air Group
RYAAY Ryanair Holdings Plc
SAVE  Spirit Airlines
SKYW  Sky West
UAL   United Continental Holdings
VLRS  Controladora Vuela Compañía de Aviación SAB de CV
ZNH   China Southern Airlines Co. Ltd.

:fresh> .s airlines

:airlines>.u
```

```
top> .l
20 airlines
top>
```

Mug Commands and Functions

Top-Level Mug Commands

.h	Provides the basic help message from the top level. Note alternatively that when **.h** is employed while Mug is an a state focused on a single company ticker, the help message includes a list of commands relevant to that specific company, or to a general company focus environment.
.doc	Opens Mug documentation in a new browser window. The documentation is in pdf format.
.h cmd	Provides a list of generic-ticker-specific commands. These commands can be used at the top> prompt assuming they come after a ticker symbol (not preceded by a "**.**") To obtain commands specific to a ticker, use the **h** function after the ticker name. The following is an example for finding `IBM` specific commands. `top> IBM h`
.sl `<cname>`	This symbol lookup service is partially functional. It is currently case-sensitive and does not have a robust matching mechanism, but it is still surprisingly useful.
.noisy	This command tells Mug to let you know of data usage that may cost points. This request can be reversed using **.quiet**.
.quiet	This command tells Mug to silence reports of data usage that may cost points. This request can be reversed using **.noisy**.
.w `low high`	This command creates a set of companies that have market caps within the range between low and high. Low and high are expressed in millions. The result is automatically put into the set called "`fresh`" destroying any data that was previously in the "`fresh`"

set. Here we see that the "fresh" set is volatile and we can not rely on it being in the state of interest. It is there for recommended to provide names to sets that will be used again using the **.s** command.

.b name name2… This command is similar to the window command regarding its ability to create a new set of companies from scratch. And this new created set can be found in the "fresh" set. Any data in the "fresh" set prior to execution of the **.s** command will be overwritten. So its important to save the fresh set first if you are concerned that data of interest may otherwise be overwritten, or that it will take a lot of work to recreate the set and you want to have the opportunity to use it later without recreating it.

.a name name2… This command is used to add a company to an existing set, or to the "fresh" set, whichever set is in focus at the time of the command. This command should thus be used only in a set is in focus, but also works at the top level regarding the most recent set that has been in focus. The syntax is basically **.a** followed by a space and then any number of ticker symbols. Companies associated with these ticker symbols will be added to the set in focus, be that "fresh" or otherwise. Since that action changes the set of companies in focus (via addition), if the focus was on a named group prior to the execution of the **.a** command, the named set does not change, but the set in focus becomes the set with name "fresh" and that can be saved to the prior name via the **.s** command if desired.

.d name This command is similar to the **.a** command, but currently only works for a single security. Thus if you want to delete multiple companies from the set in focus, you are currently required to delete each one individually with separate **.d** commands.

<name> Mug interprets any command phrase that does not begin with the "**.**" Character to begin with the ticker name of a company or alternatively begin with the

name of a named set of companies (created by a prior application of the **.s** command). If the name is identical to a ticker, then Mug transforms its state into one focused on the specific ticker rather than mapping functions on a group (or set) of tickers. When in this state, ticker specific commands are used at the command line, and a "**.**" Symbol is required to be the initial character of each command.

`<ticker>` **h** — This command provides the same ticker-specific and ticker-focus general commands, that would be provided as if we executed the **.h** command while having `<ticker>` in focus.

`<ticker> <cmd>` — This provides the same evaluation of a command with respect to a company ticker symbol as if the command were to be executed with the company ticker in focus. This command phrase provides the ability to get detailed information on a ticker of interest, which not being required to change the focus of the current set (or alternative ticker).

.l — This command list named sets and pairs when at the top level. Note that this command behaves differently when Mug's focus is not at the top layer of the application. For example, when Mug's focus is on a named set (including "`fresh`"), the **.l** command will list the company tickers within the current set in focus, rather than listing all the named sets (as would happen if the focus were at `top`).

.p `<t> <t>` — Create un-named pair (not yet implemented)

`<name>` — This command places Mug's focus on a specific company ticker if `<name>` happens to be identical to any ticker. If this is not the case, and if `<name>` is alternatively the name of a group of tickers, then the command will put that group (set of company tickers) into focus. If the prior focus was "`fresh`", its contents will be retained, and thus fresh can be put back into focus by simply typing **fresh**. Keep in mind that the

	"`fresh`" set is volatile, and subject to deletion when commands such as **`.w`**, **`.a`**, and **`.d`** are used.
`.q`	This command is used to quit the Mug session. (Advanced topic: The prior mug session can be returned to if Mug was called using the (-main) function from a clojure repl prompt.)

Company-Level Mug Commands

`.u`	Up. This command returns the focus from the company to the set that was previously in focus or to the top level if that was the most recent focus prior to a focus on the specific company.
`.cname`	Company-name.
`.i`	Industry. There are specific industry classifications, and the classification to which the ticker is a member is obtained with this command.
`.web`	Website. This command obtains the main website of the company of interest. The website opens in your system's default web browser. If a browser window is already open, the company website may appear in a new tab of the existing window rather than a new independent window.
`.desc`	This command provides a concise description of the company.
`.ceo`	Obtain name of the company's CEO.
`.s`	Sector. There are specific sector classifications, and the classification to which the ticker is a member is obtained with this command.
`.emp`	Obtain company's employee count.
`.zipcode`	This zipcode is scraped from the sec website. For this command to successfully return a valid zipcode, the company must have an SEC (U. S. Securities and

	Exchange Commission) filings page searchable by the company's ticker.
.oweb	Opens the company's main web homepage in the default browser.
.yahoo	Opens the yahoo-finance website associated with the company ticker.
.sec	Opens the SEC (U. S. Securities and Exchange Commission) website containing the most recent filings for the company of interest.
.so	Shares Outstanding.
.pf	Number of shares publically traded.
.p	Current share price.
.v	The daily volume history averaged over the past 30 days is multiplied by the current share price, so that volume would be expresses in units that can be compared among companies.
.b	CAPM beta. Correlation of company returns to that of the entire market, or simply β as defined in the Capital Asset Pricing Model.
.mkt	Marketcap. Inferred total equity trading value of the company. Current share price times shares outstanding.
.c	Cash. Most recent cash balance from financial statements.
.d	Debt. Most recent total debt from financial statements.
.r	Revenue. Most recent revenue amount from financial statements.

.g	Gross profit. Most recent gross profit amount from financial statements.
.e	EBITDA. Most recent EBITDA amount from financial statements. Proxy for available cash flow. Used by banks for analyzing debt coverage.
.d2e	Deb-to-Equity. Most recent ratio from financial statements.
.ir	Insider-Roster. List of company insiders.
.db	Deep-Book. Exchange quote information.
.sp	Splits. History of stock splits.
.fo	Fund-Ownership. Funds with a position in the company.
.cffd	Total financing cash flow by quarter as reported. Useful for identifying company stock offerings.
.cff	The total of the absolute value of the cash flows reported by the **.cffd** command. Note that the **.cff** command calls **.cffd** in certain circumstances for convenience.
.io	Institutional-Ownership. Non-fund institutional entities that own equity in the company.
.cc	Ceo-Compensation.