

Shim Command Reference

R P Herrold

Last revised: May 22, 2008

R P Herrold
Post office box 12069
Columbus, Ohio, 43212, USA
Phone: 614-488-6954
E-mail: info@owlriver.com
Web site: www.owlriver.com

All rights reserved
© 2007 by R P Herrold

No part of the material protected by this copyright notice may be reproduced or utilized in any form or by any means – graphic, electronic, or mechanical, including photocopying, taping, recording or by any other information storage and retrieval system, without prior, written permission from R P Herrold.

Contents

Table of Contents	xi
Disclaimer	xiii
License	xv
To Do	xv

I Introduction 1

1 Introduction 3

1.1 Trademarks	3
1.2 Quotation of Copyrighted material	4
1.3 Disclaimer of the Author and Publisher	4
1.4 No Warranties, express or implied	4
1.5 Typographic conventions	4
1.6 How this document has been compiled	5

II The commands, and their syntax 13

2 Introduction to command verbs 15

2.1 Description - command verbs	15
2.2 Line wrapped output	15

3 The commands, alphabetically 21

3.1 account - get account quads	22
3.1.1 Description	22
3.1.2 Peers	22
3.2 acct - get account quads	23
3.2.1 Description	23
3.2.2 Usage	23
3.2.3 Peers	25
3.2.4 Listing of: help acct	26
3.3 bind - FIXME	27
3.3.1 Description	27

3.3.2	Usage	27
3.4	book - subscribe to market depth	28
3.4.1	Description	28
3.4.2	Usage	28
3.4.3	See related	29
3.4.4	Peers	29
3.4.5	Listing of: help book	30
3.5	cash - FIXME	31
3.5.1	Description	31
3.5.2	Usage	31
3.5.3	Peers	31
3.6	dbms - describe the dbms to use	32
3.6.1	Description	32
3.6.2	Usage	32
3.6.3	Peers	32
5.1.5	Listing of: help link	103
3.7	exec - get execution log FIXME	34
3.7.1	Description	34
3.7.2	Usage	34
3.7.3	See related	34
3.8	exercise - FIXME	35
3.8.1	Description	35
3.8.2	Usage	35
3.8.3	Peers	35
3.9	feed - describe the upstream TWS market data feed parameters	36
3.9.1	Usage	36
3.9.2	Peers	36
3.9.3	Antecedents	36
5.1.5	Listing of: help link	103
3.10	help - command verb help	38
3.10.1	Description	38
3.10.2	Usage	38
5.1.2	Listing of: help help	100
3.11	history - ask history query	40
3.11.1	Description	40
3.11.2	Peers	40
3.12	info - get contract info	41
3.12.1	Description	41
3.12.2	Usage	41
3.12.3	Peers	41
3.12.4	Listing of: help info	42
3.13	list - list subscriptions	43

3.13.1	Description	43
3.13.2	Usage	43
3.13.3	Listing of: help list	44
3.14	load - Read, or re-read SubRequest table	45
3.14.1	Description	45
3.14.2	Usage	45
3.14.3	Peers	46
3.14.4	Antecedents	46
3.14.5	Listing of: help load	47
3.15	news - control bulletins	48
3.15.1	Description	48
3.15.2	Usage	48
3.15.3	Limitation	48
3.15.4	See related	48
3.15.5	Listing of: help news	49
3.16	next - ping the TWS	50
3.16.1	Description	50
3.16.2	Usage	50
3.16.3	Peers	50
3.16.4	Listing of: help next	51
3.17	open - check open orders	52
3.17.1	Description	52
3.17.2	Usage	52
3.17.3	Listing of: help open	53
3.18	order - manage a LINEITEM	54
3.18.1	Description	54
3.18.2	Usage	54
3.18.3	Peers	54
3.19	past - ask history query	55
3.19.1	Description	55
3.19.2	Quick lookup of PastFilter configuration id's	60
3.19.3	Usage	62
3.19.4	Extended example	63
3.19.5	Peers	70
3.19.6	Listing of: help past	71
3.20	ping - log time, comment through EOL	72
3.20.1	Description	72
3.20.2	Usage	72
3.20.3	Peers	72
3.20.4	Listing of: help ping	73
3.21	quit - exit the program	74
3.21.1	Description	74

3.21.2	Usage	74
3.21.3	Listing of: <code>help quit</code>	75
3.22	<code>read</code> - FIXME	76
3.22.1	Description	76
3.22.2	Usage	76
3.22.3	Peers	76
3.22.4	Antecedents	76
3.22.5	Listing of: <code>help read</code>	77
3.23	<code>scan</code> - event scanner TBD	78
3.23.1	Description	78
3.23.2	Usage	78
3.23.3	Peers	78
3.24	<code>tick</code> - subscribe mk data	79
3.24.1	Description	79
3.24.2	Usage	79
3.24.3	Peers	80
3.24.4	Future explanation	80
3.24.5	Listing of: <code>help tick</code>	82
3.25	<code>transmit</code> - FIXME	83
3.25.1	Description	83
3.25.2	Usage	83
3.26	<code>verb</code> - set tws log level	84
3.26.1	Description	84
3.26.2	Usage	84
3.26.3	Listing of: <code>help verb</code>	85
3.27	<code>wait</code> - sleep shim N secs	86
3.27.1	Description	86
3.27.2	Usage	86
3.27.3	Listing of: <code>help wait</code>	87
3.28	<code>wake</code> - clear pause count	88
3.28.1	Description	88
3.28.2	Usage	88
3.28.3	Listing of: <code>help wake</code>	90
3.29	<code>wild</code> - abstract contract	91
3.29.1	Description	91
3.29.2	Usage	91
3.29.3	Peers	91
3.30	<code>wire</code> - accumulate orders	92
3.30.1	Description	92
3.30.2	Peers	92
3.30.3	Listing of: <code>help wire</code>	93
3.31	<code>xmit</code> - release tws order	94

3.31.1	Description	94
3.31.2	Usage	94
3.31.3	See related	94
4	Parameters, common to the command verbs	95
4.1	Parameters to the command verbs	96
4.1.1	Simple parameters	96
4.1.2	Order (wire) parameters	96
5	'shim -help' matters	97
5.1	-help - short form help from the program	98
5.1.1	Description	98
5.1.2	Listing of: help help	100
5.1.3	Listing of: help args	101
5.1.4	Listing of: help cmds	102
5.1.5	Listing of: help link	103
5.2	shim Modes	104
5.2.1	Listing of: help mode	105
5.3	shim Options	106
5.3.1	Listing of: help opts	107
5.4	.shimrc - optional file to describe shim parameters	108
5.4.1	Usage	108
5.4.2	Peers	109
6	Numbering - Commands, Requests, Messages, Comments	111
6.1	Overview on message numbering	111
6.2	message class, message value and message version	112
6.2.1	message class	113
6.2.2	message value	113
6.2.3	message version	113
6.2.4	TWS message value and message version co-ordination	113
6.3	message values in the TWS	114
6.4	Java sample client	114
6.4.1	Rationale' for consulting the Java sample client	115
6.4.2	How to view a permanent page URL on the IB site	115
6.4.3	Retrieving the Java sample client	116
6.5	Numbering in the Java sample client	117
6.5.1	Numbering of Requests in EClientSocket.java	117
6.5.2	Numbering of Messages in EReader.java	118
6.5.3	Numbering of Tick Types in TickType.java	119
6.6	Numbering in rule.c of the shim	120
6.6.1	Numbering of Commands in rule.c	120
6.6.2	Numbering of Requests in rule.c	121

6.6.3	Numbering of Messages in <code>rule.c</code>	122
6.6.4	Numbering of Comments in <code>rule.c</code>	122

III Guided Tutorial 123

7 A Tour of Tables 127

7.1	Tables	128
7.1.1	Why so many tables	128
7.1.2	What tables are there anyway?	128
7.1.3	Which tables are safe to alter	129
7.1.4	Adding additional tables	130
7.1.5	The initial database load process	131
7.1.6	Each starts with the initial database load process	132

8 Working with the database 135

8.1	The shim database and <code>CONTRACT</code> IDs	136
8.1.1	Looking up an underlying <code>SYMBOL</code> from the <code>cid</code> - step by step	136
8.1.2	Looking up a underlying <code>SYMBOL</code> from the <code>cid</code> - with <code>LEFT JOIN</code>	142
8.1.3	Looking up a <code>CONTRACT.UID</code> with <code>LEFT JOIN</code>	146
8.1.4	Adding a new underlying <code>SYMBOL</code> to the <code>CONTRACT</code> table	148
8.1.5	Fixing the <code>make test</code>	151
8.1.6	Tabular database table listings in other contexts	154
8.1.7	How to extend the <code>Symbol</code> (and then <code>CONTRACT</code>) tables	158
8.1.8	Bulk loading the <code>CONTRACT</code> table	168

9 Commands and the database together 169

9.1	Market Data, History, and Market Depth	170
9.2	Market Data subscription	171
9.2.1	Subscribing to Market Data	171
9.2.2	Unsubscribing from Market Data	172
9.3	History retrieval	172
9.3.1	Retrieving a History set – one off current	174
9.3.2	Retrieving a History set – recurring current	175
9.3.3	History Pacing Violations	175
9.4	Market Depth subscription	178
9.4.1	Subscribing to Market Depth	178
9.4.2	Unsubscribing from Market Depth	179

10 Adding a web browser interface	181
10.1 Look up interface	182
IV Preparing this document	185
11 Preparing this document	187
12 The writing process	189
12.1 Adding new commands	189
12.2 Editing prior text	190
V Conclusion	191
Appendix	195
Bibliography	198
Index	198

Disclaimer

DISCLAIMER: This documentation is presently under active development and as such there may be mistakes and omissions – watch out for these and please report any you find to the mailing list,

`ts-general@trading-shim.org`

or by a private email to the author at the email address indicated on the Copyright page.

The latest version is available on-line. Contributions of material, suggestions and corrections are welcome.

License

The `trading-shim` is an open-source project. Most portions of the `trading-shim` are licensed for copying under the terms of the GNU General Public License, version 3. Please refer to the `COPYING` file for details.

As noted at the website, as the sole copyright holder or assignee, and exclusive licensor of the `trading-shim`, `trading-shim.com`, LLC has and offers the option of sub-licensing its software under alternative commercial terms.

Reminders to the author

To Do

Update as of 24 October 2007:

Sections still to write (in addition to completing the per command summaries) include a section about:

1. How the shim exposes only three major types of command, and hides the TWS' enormous complexity: global state commands, 'cid' specific commands, and the 'order' command
2. writing the 'order' tutorial (I have this blocked out, however, and need to describe bracket and OCA orders with worked examples)
3. completion of a bit more on load.sql, and discussing strategies for migrating data as the underlying database schema may need to change, or be unloaded and re-loaded
4. reorganizing some code back into appendices, and moving some narrative back and around in the tutorial

Please note: This is a draft, and subject to major revision; large parts are known to be stale and not recently tested.

Part I
Introduction

Chapter 1

Introduction

This manual relates to the trading-shimTM which is a command-line and dbms controlled interface to the socket-based API of Interactive Brokers' Trader Workstation.

We describe the syntax for each command, and provide an alphabetical quick reference. Applied examples of useful command sequences are saved for later. We also explore topics related to working with the database used by the shim. This includes pointing out the purposes of some tables which may sensibly be used by an end user, usually on a 'read only' basis to give human readable context to values used by the shim, or in some cases, to point out ways a user may safely add additional or local values to the population of the tables as provided by the developers.

Please note that in this work, we adopt a more conversational style of writing than the formal 'The Trading-Shim Manual', authored by Bill Pippin. Neither is inherently 'better' than the other, but rather are targetted to different audience needs.

1.1 Trademarks

Generally, the appearance of trademarks or registered trademarks within this work are done as a nominative and factual matter, as and for description and identification. See, generally, 15 USC 1115(b)(4). We are in no wise interested in any implied trademark infringement or counterfeiting (11 USC 1114(1)); false designation or unfair competition (15 USC 1125(a)); dilution (15 USC 1125(c); common law infringement or unfair competition, or dilution; violation of business practice law or regulation as to use of marks.

The uses of **trading-shim** and the short form **shim** are intentionally noted here and not later capitalized. For marks other than those held by **trading-shim.com, LLC**, we strive to note this status with Capitalization marking.

1.2 Quotation of Copyrighted material

We quote from the shim source code, and from sessions running the shim, both in examples, and more extensively, in producing a listing of all shim shim entries. As the shim help content was authored by Bill Pippin, we acknowledge and thank him for production of this content. We refresh it with an mechanical script from time to time.

1.3 Disclaimer of the Author and Publisher

”This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in rendering legal, tax, accounting, or other professional service. If legal advice or other expert assistance is required, the services of a competent professional person should be sought.”

– from a Declaration of Principles jointly adopted by a Committee of the American Bar Association and a Committee of Publishers and Associations.

1.4 No Warranties, express or implied

Absent a prior, formal, written, paid up and commercial license and support contract, there is **no warranty, expressed or implied, nor guarantee against any sort of perceived adverse result, regardless any prior contrary request for assurance; no person except the officers of trading-shim.com, LLC** may in any fashion vary this term, except that it be by a writing countersigned by at least two such officers of trading-shim.com, LLC.

1.5 Typographic conventions

Most narrative text is simply in regular Times Roman.

A code listing or screen scrape is usually wrapped into a standalone code block, set up in monospace font:

```
[herrold@centos-4 docs]$ ls -l ../shim-071016 | cut -c 24-79 | head
herrold      4096 Oct 16 17:34 bin
herrold     35147 Jul  9 15:16 COPYING
herrold      4096 Oct 16 16:04 dep
herrold      4096 Oct 16 17:41 doc
herrold       936 Oct 16 17:38 FUT.SMART.YM.hql
herrold     17908 Jul  9 15:13 INSTALL
```

```
herrold      47 Oct 16 17:38 keying
herrold      4096 Oct 16 16:04 lib
herrold      4096 Oct 16 16:04 log
[herrold@centos-4 docs]$
```

Unix commands, Linux commands, filesystem paths and scripts, and shim commands are set off in a san-serif font when not in a verbatim code block as well: `grep`, `bin/includes`, `info`, `past` and `tick`.

MySQL commands are set off in an ALL CAPS san-serif font: `INSERT`, `LEFT JOIN`, and `SELECT`.

MySQL table names, and table names with a specific field were previously indicated: ‘Contract’, and ‘Contract.uid’ They were changed to a SMALL CAPS font: `CONTRACT` and `CONTRACT.UID`

1.6 How this document has been compiled

The shim has been noted as an explorational project, and so has evolved over time; commands, Options, and such have come and gone as our understanding of the subject domain has changed. The commands and options been added, renamed, and grown or changed in meaning over time, and so forth. As the source code contines to grow, this document may fall out of date and need to be re-synchronized against the one true point of authority: the source code itself.

This reference is broken into certain major parts:

1. the statement of the Syntax of each command, sorted alphabetically by first command word [updated through 09 July 2007]. Presently, some commands are explained lightly, if at all; others in a uniform form, and a few with local examples. This variation will probably be re-organized away in later revisions of this draft.
2. a Guided Tutorial to first then database, and then common command usages
3. a Section of Troubleshooting commonly observed issues and error messages
4. We manually drill in additional index entries and cross references as well.

The list of commands which we discuss in the first part was built by scanning the entries in `rule.c`, [formerly `tabs.c`] and using the Unix TM derived `grep` command to view the relevant lines for the names of the potential commands to document.

As of the 12 Oct 2007 revision, this yields this cluttered list:

```
[herrold@centos-4 shim]$ ./docs/get_cmd_list.sh shim_071011 | grep cmd
new (p) cmd::Help(w, STV( 1), c, T0x, "help", null),// log time, comment
new (p) cmd::Ping(w, STV( 2), c, T01, "ping", null),// ping connectivity
new (p) cmd::Next(w, STV( 3), c, T00, "next", null),// tick, order index
new (p) cmd::List(w, STV( 4), c, T00, "list", null),// list subscriptions
new (p) cmd::Wait(w, STV( 5), c, T02, "wait", null),// sleep shim n secs
new (p) cmd::Wake(w, STV( 6), c, T00, "wake", null),// clear pause count
new (p) cmd::Quit(w, STV( 7), c, T00, "quit", null),// that's it for now
new (p) cmd::Verb(w, STV(11), c, T06, "verb", xact),// set tws log level
new (p) cmd::News(w, STV(12), c, T05, "news", news),// control bulletins
new (p) cmd::Open(w, STV(13), c, T00, "open", xact),// check open orders
new (p) cmd::Acct(w, STV(14), c, T03, "acct", acct),// get account quads
new (p) cmd::Info(w, STV(15), c, T07, "info", data),// req contract data
new (p) cmd::Wild(w, STV(16), c, T08, "wild", data),// abstract contract
new (p) cmd::Tick(w, STV(17), c, T13, "tick", tick),// subscribe mk data
new (p) cmd::Book(w, STV(18), c, T13, "book", book),// also market depth
new (p) cmd::Past(w, STV(19), c, T14, "past", past),// ask history query
new (p) cmd::Scan(w, STV(20), c, T13, "scan", news),// event scanner TBD
new (p) cmd::Exec(w, STV(21), c, T02, "exec", xact),// get execution log
new (p) cmd::Read(w, STV(22), c, T00, "read", null),// append new tuples
new (p) cmd::Load(w, STV(23), c, T00, "load", tick),// reread subrequest
new (p) cmd::Bind(w, STV(25), c, T11, "bind", null),// bind sym to tuple
new (p) cmd::Wire(w, STV(26), c, T16, "wire", xact),// accumulate orders
new (p) cmd::Xmit(w, STV(27), c, T04, "xmit", xact),// release tws order
new (p) cmd::Cash(w, STV(28), c, T15, "cash", xact),// use options right
new (p) cmd::Acct(w, STV(14), c, T03, "account", acct),
new (p) cmd::Past(w, STV(19), c, T13, "history", past),
new (p) cmd::Wire(w, STV(26), c, T16, "order", xact),
new (p) cmd::Xmit(w, STV(27), c, T16, "transmit", xact),
new (p) cmd::Cash(w, STV(28), c, T15, "exercise", xact)
new (p) cmd::Dbms(w, STV( 1), c, T0, "dbms", null), // database connection
new (p) cmd::Feed(w, STV( 2), c, T1, "feed", null) // upstream tws params
[herrold@centos-4 shim]$ date
Fri Oct 12 13:32:55 EDT 2007
[herrold@centos-4 shim]$
```

which, is just too cluttered to be very useful. With a bit more pipeline filtering code, we can massage that mess into a more useful checklist of commands:

```
[herrold@centos-4 shim]$ date ; ./docs/get_cmd_list.sh shim_071011 | \
```

```

    grep cmd | awk -F'"' {'print $2'} | sort | tr '\n' ' ' | fmt -t
Fri Oct 12 13:38:29 EDT 2007
account acct bind book cash dbms exec exercise feed help history info
    list load news next open order past ping quit read scan tick transmit
    verb wait wake wild wire xmit
[herrold@centos-4 shim]$

```

Similarly useful in building other parts of the documentation is using variations of:

```
grep '.dual' syms.c
```

to draw out a list of program options, RC file Key-value name labels, and runtime Options and aliases, among other data 'constants'.

This produces a checklist of the shim program's options:

```

[herrold@centos-4 shim]$ date ; grep '.dual' shim_071011/src/syms.c | \
    grep 'Option'| awk -F'"' {'print $2'} | sort | tr '\n' ' ' | fmt -t
Fri Oct 12 13:42:25 EDT 2007
cmds cout fast file init load logd many null opts pane save stdout
    syslog window
[herrold@centos-4 shim]$

```

This produces a checklist of the shim help assistance topics:

```

[herrold@centos-4 shim]$ date ; grep '.dual' shim_071011/src/syms.c | \
    grep 'Assist'| awk -F'"' {'print $2'} | sort | tr '\n' ' ' | fmt -t
Fri Oct 12 14:24:43 EDT 2007
acct args bind book cash cmds exec help info link list load mode news next
    open opts past ping quit read scan tick verb wait wake wild wire xmit
[herrold@centos-4 shim]$

```

This produces a enumeration checklist of binary argument forms:

```

[herrold@centos-4 shim]$ date ; grep '.dual' shim_071011/src/syms.c | \
    egrep "(False|True)" | awk -F'"' {'print $2'} | sort | tr '\n' ' ' | \
    fmt -t
Fri Oct 12 14:28:46 EDT 2007
add all del new no off on start stop yes

```

We see also the modes: data and risk, and some Equality forms: =, eq, as, and to.

Stub parts from shim -help

FIXME
shim help
help help;

Online Help

The help command provides information about trading-shim operation and the shim command set. There is one argument, selecting a command verb or topic:

The shim command verbs:

help
ping next
wait wake quit
read load list
*bind
verb news open acct
info *wild *exec
tick book past *scan
wire *xmit *cash

General topics:

mode
opts
link
cmds
args

Note: commands marked
with an asterisk are
not yet implemented.

Syntax:

```
help <verb>;  
help <topic>;
```

where <verb> or <topic> is from one of the tables above.

snapshotted 19 Oct 2007

FIXME
shim modes

Modes:

```
--help      # list the help command arguments and accept help commands

# real modes, requiring access to an IB tws:
--data      # process subscriptions and log resulting tick stream events
--risk      # accept full command set, send requests, and log all events

# test modes, with no connection to the tws:
--play      # read events from the image file and send text to stdout
--unit      # for internal use; unstable though otherwise harmless
```

FIXME

shim options

1. cmds
2. cout
3. fast
4. file
5. init
6. load
7. logd
8. many
9. null
10. opts
11. pane
12. save
13. stdout
14. syslog
15. window

this is to some degree a classified list, ordered by FIXME describe.

The shim command verbs:

General topics:

```
-----  
  
help  
ping    next  
wait    wake    quit  
read    load    list  
*bind  
verb    news    open    acct  
info    *wild   *exec  
tick    book    past    *scan  
wire    *xmit   *cash
```

Note: commands marked with an asterisk are not yet implemented.

FIXME
shim RC file

Part II

The commands, and their syntax

Chapter 2

Introduction to command verbs

2.1 Description - command verbs

This section will contain more descriptive material about the command verbs

2.2 Line wrapped output

Some output here is too wide for a conventional 80 character line; to accommodate this, consider the following sample line:

```
Feb  5 21:15:46 centos-4 : shim|data|0.28| 2433|76546|   1382844|  
      3| 4| 2|      -1|2107|HMDS data farm connection is inactive  
      but should be available upon demand.:ushmds2a|
```

which in this example is broken after the time counter (maintained by the shim), but before the Reply message tuple: 3— 4— 2—. A counted eight spaces are inserted, for \TeX layout reasons inside the ‘verbatim’ section.

A very long response, such as the final message field shown, may also be broken later at a convenient whitespace. By and large this permits retaining whitespace within ‘pipe’ separators. Sometimes we move the ‘pipe’ to the continuation line, as on occasion there is leading white space padding which we may wish to see, but that would otherwise be lost.

Note that the examples in the reference have come from many shim versions over time, and reflect debugging examples for code which may not be in the present testing scripts, or expose output log formats which are not presently used. The development of the shim has changed such formats over time, and upon request, we will add a note that a given example is of historical or pedigological interest, rather than something which may be produced with the then-current day’s release.

Type	Source	Tag	Version
Command	1	see <code>src/rule.c</code>	
Request	2	see <code>src/rule.c</code>	-few-
Message	3	FIXME	-several-
Comment	4	-none-	

We mentioned a ‘tuple’ above, will refer to it again from time to time, and see it in nearly all examples; a tuple consists of three parts: the Source, the message Tag, and the tag Version. This permits separating Command, Request, Message, and Comment entries from one another, and within a given ‘Source’, the sub-element provided, by its ‘Tag’, and finally, as is occasionally needed as a message format may change, the particular ‘Version’ of that ‘Tag’

- Command: is the shim command text language used by a client to the shim. We see them enumerated thus:

```
[herrold@centos-5 src]$ grep TagName rule.c | grep 'STV(1,' | \
awk '{print $7" "$5}' | sort -n | grep ^[0-9]
1), "help"),
2), "ping"),
3), "next"),
...
27), "xmit"),
28), "cash"),
28), "exercise"),
[herrold@centos-5 src]$
```

From a test run, we might example the file ‘cmdinput.txt’:

```
[herrold@centos-5 shim_071221]$ cat cmdinput.txt
order(3,LMT,Create,2,60.0,0.0,0);
order(4,LMT,Create,2,70.0,0.0,0);
order(3,LMT,Submit,2,60.0,0.0,0);
open;
order(3,MKT,Submit,2,00.0,0.0,0);
order(4,MKT,Submit,2,00.0,0.0,0);
FIXME past add 181 6 now;
select next;
select news all;
cancel news all;
select acct;
```

```

select book add 15 3;
select book add 178 7;
load;
quit;
[herrold@centos-5 shim_071221]$

```

The rest of the example is pulled from the matching 'ShimText' file. In part it reads:

```

11304|45056| 1031692|4|100| 0|# |4|100|0|*****|
11304|45056| 1031699|4|101| 0|# |4|101|0|0.52|070831|risk|
11304|45056| 1031703|4|100| 0|# |4|100|0|*****|
11304|45056| 1031682|4|102| 0|# |4|102|0|23|11304|39|20071226
      12:30:55 EST|Connect with: cv 23, id 11304, sv 39|
11304|45056| 1177737|3| 9| 1|1|
11304|45056| 1194419|3| 4| 2|      -1|2104|Market data farm
      connection is OK:usfarm|
11304|45056| 1194434|3| 4| 2|      -1|2104|Market data farm
      connection is OK:usfuture|
11304|45056| 1194452|3| 4| 2|      -1|2107|HMDS data farm
      connection is inactive but should be available upon
      demand.ushmds2a|
11304|45059| 3990653|1|26| 0|order(3,LMT,Create,2,60.0,0.0,0);|
11304|45059| 4010547|2| 3|15|1|      3|LMT|BUY|STK.SMART.AIG.
11304|45060| 4990331|1|26| 0|order(4,LMT,Create,2,70.0,0.0,0);|
11304|45060| 5010145|2| 3|15|2|      4|LMT|SELL|STK.SMART.AIG.
11304|45061| 5994359|1|26| 0|order(3,LMT,Submit,2,60.0,0.0,0);|
11304|45061| 6014113|2| 3|15|1|      3|LMT|BUY|STK.SMART.AIG.
11304|45061| 6122540|3|11| 4|1|AIG|STK|||SMART|USD|AIG|
      00018037.44913a80.01.01|20071226 12:31:01|DU10126|
      ISLAND|BOT|2|59.26|1671595100|11304|0|
11304|45061| 6123046|3| 5|10|1|AIG|STK||0.0|?|SMART|USD|
      AIG|BUY|2|LMT|60.0|0.0|GTC||DU10126|C|0||11304|1671595100
      |false|false|0||1671595100.0/DU10126/100|||||0|0 |
      |||||false|false|false|false||3|false|false||0|1|

```

- Request: is the binary translation, occasionally using information from the database, of a Command to a form which the upstream TWS receives across a socket connection.

```

[herrold@centos-5 src]$ grep TagName rule.c | grep 'STV(2,' | \
awk '{print $9" "$6}' | sort -n | grep ^[0-9]

```

```

1, "ReqMktData"
2, "EndMktData"
...
23, "EndScanSub"
25, "EndHistory"
[herrold@centos-5 src]$

```

- Message: is a binary format which the TWS sends across a socket connection, to transfer both control state and substantive content to a downstream client.

Some messages were received on an unsolicited basis from the upstream TWS as to the connection status and availability of resources 'upstream' of it:

```

11304|45056| 1177737|3| 9| 1|1|
11304|45056| 1194419|3| 4| 2| -1|2104|Market data farm
connection is OK:usfarm|
11304|45056| 1194434|3| 4| 2| -1|2104|Market data farm
connection is OK:usfuture|
11304|45056| 1194452|3| 4| 2| -1|2107|HMDS data farm
connection is inactive but should be available upon
demand.ushmds2a|

```

Other messages were in reply to Commands, translated into Requests. This Created, but did not release to be Submit[ed] upstream, an order based on LINEITEM.UID: 3

```

11304|45059| 3990653|1|26| 0|order(3,LMT,Create,2,60.0,0.0,0);|
11304|45059| 4010547|2| 3|15|1| 3|LMT|BUY|STK.SMART.AIG.

```

A few seconds later, the order is Submit[ed] upstream, and fills at once, as it is actually already past the limit price stated:

```

11304|45061| 5994359|1|26| 0|order(3,LMT,Submit,2,60.0,0.0,0);|
11304|45061| 6014113|2| 3|15|1| 3|LMT|BUY|STK.SMART.AIG.
11304|45061| 6122540|3|11| 4|1|AIG|STK| |||SMART|USD|AIG|
00018037.44913a80.01.01|20071226 12:31:01|DU10126|
ISLAND|BOT|2|59.26|1671595100|11304|0|

```

QUERY: How can we tell the entry at 6014113 from the earlier one at 4010547 – each line appears identical, although one is a Create, and the second a Submit?

- Comment: additional text added by the shim locally toward its output consumer, expanding on its state.

```
11304|45056| 1031692|4|100| 0|# |4|100|0|*****|
11304|45056| 1031699|4|101| 0|# |4|101|0|0.52|070831|risk|
11304|45056| 1031703|4|100| 0|# |4|100|0|*****|
11304|45056| 1031682|4|102| 0|# |4|102|0|23|11304|39|20071226
```

which describe the state of the connection, and message versions which the shim used during its connection to the TWS.

Chapter 3

The commands, alphabetically

A recent re-work of the command language has changed it to a ‘verb object’ command form, which relies heavily on the verbs: `select` and `cancel`. Formerly, and throughout this work until revisions are complete, one might see the follow form:

```
tick add 181 1;  
tick del 181 1;
```

which is obsolete, and now carries the forms, respectively:

```
select tick 181 1;  
cancel tick 181 1;
```

This change in expression applies to the following commands. A back link to this discussion has been placed at the top of each affected command listed below. As each command’ section is revised, the backlink will be removed.

- `acct` amended
- `book` not fixed
- `exec` not fixed
- `info` amended
- `news` amended
- `next` amended
- `past` not fixed
- `tick` amended

3.1 account - get account quads

3.1.1 Description

get account quads

3.1.2 Peers

It has a synonym called `acct`; see `acct` (at: [3.2](#)) which is also used. `acct` is preferred in `help` system documentation matters.

3.2 acct - get account quads

3.2.1 Description

get account quads

```
select acct;
```

3.2.2 Usage

Minimal usage:

Example from bin/includes:

```
select acct;  
quit;
```

(From bin/includes)

produces in the logfile:

```

21429|46333| 1030324|4|100| 0|# |4|100|0|*****|
21429|46333| 1030329|4|101| 0|# |4|101|0|0.81|070831|data|
21429|46333| 1030332|4|100| 0|# |4|100|0|*****|
21429|46333| 1030318|4|102| 0|# |4|102|0|23|1|40|20080512 12:52:12
      EST|Connect with: cv 23, id 1, sv 40|
21429|46333| 4445235|3| 9| 1|1|
21429|46336| 4465473|3| 4| 2|      -1|2104|Market data farm connection is
      OK:usfarm|
select acct;
21429|46341| 9821225|1| 3| 0|select acct;|
21429|46341| 9821262|2| 6| 2|true|
21429|46341| 9828746|3| 6| 2|AccountCode      | DU10126|
      |DU10126|
21429|46341| 9832393|3| 6| 2|AccountReady      |      true|
      |DU10126|
21429|46341| 9833466|3| 6| 2|AccountType      |UNIVERSAL|
      |DU10126|
21429|46341| 9834409|3| 6| 2|AccruedCash      |
      50.80|BASE|DU10126|
21429|46341| 9835303|3| 6| 2|AccruedCash      |
      0.00|EUR |DU10126|
21429|46341| 9836205|3| 6| 2|AccruedCash      |
      50.80|USD |DU10126|
21429|46341| 9837335|3| 6| 2|AccruedCash-C      |
      0.00|USD |DU10126|
21429|46341| 9838370|3| 6| 2|AccruedCash-S      |
      50.80|USD |DU10126|
21429|46341| 9839344|3| 6| 2|AvailableFunds      |
      |186304.23|USD |DU10126|
21429|46341| 9840768|3| 6| 2|AvailableFunds-C      |
      0.00|USD |DU10126|
21429|46341| 9841763|3| 6| 2|AvailableFunds-S      |
      |186304.23|USD |DU10126|
21429|46341| 9842743|3| 6| 2|BuyingPower      |
      |744440.94|USD |DU10126|
21429|46341| 9843678|3| 6| 2|CashBalance      |
      |158123.51|BASE|DU10126|
21429|46341| 9844602|3| 6| 2|CashBalance      |
      -59.46|EUR |DU10126|
21429|46341| 9845486|3| 6| 2|CashBalance      |
      |158215.93|USD |DU10126|
21429|46341| 9846378|3| 6| 2|Currency      |
      BASE|BASE|DU10126|
21429|46341| 9847625|3| 6| 2|Currency      |
      EUR|EUR |DU10126|
21429|46341| 9848837|3| 6| 2|Currency      |
      USD|USD |DU10126|
21429|46341| 9849782|3| 6| 2|Cushion      |      0.95207|

```

Note that this account has conducted trades in both USD and EUR, and so is reported in both the base currency of USD, and also the EUR component.

When the markets are open, there will be periodic updates as to account margin capacity, as well as to values of underlying positions (including Forex) as prices fluctuate. The command may be repeated from time to time, and no express limits by IB on repetition are known as documented, beyond the general 50 transactions per second upstream transaction request limit.

3.2.3 Peers

It has a synonym called `account`; see `account` (at: [3.1](#)) which is also used. `acct` is preferred in `help` system documentation matters.

3.2.4 Listing of: help acct

Refreshed from: shim-071228

3.3 bind - FIXME

FIXME - not yet implemented

3.3.1 Description

3.3.2 Usage

Minimal usage:

produces in the logfile:

3.4 book - subscribe to market depth

This form of the comand in obsolete, and pending re-write. (see: the note at the start of Cp. 3).

3.4.1 Description

subscribe to market depth – think: NYSE OpenBook

```
book Op Cid I;
```

where:

- Op : one of: add, del
- Cid : the contract id
- I : the configuration id, from table: DEPTHLIMIT. That is: the number of ‘lines’ from the ‘top’ of the book to display down from the ‘Bid/Ask’ frontier.

3.4.2 Usage

Minimal usage:

```
book add 15 3;
```

subscribes (in one dataset), to AIG market depth information
We can also then stop it thus:

```
book del 15 3;
```

TBD: The del operator is presently non-functional - 071112
(From bin/includes)

The first command produces in the logfile:

```
Nov 27 15:58:53 centos-4 : 1838|57533|3128310288|1|18| 0|book add 15 3;|
Nov 27 15:58:53 centos-4 : 1838|57533|3128329934|2|10| 3|3|15|3|
Nov 27 15:58:54 centos-4 : 1838|57534|3128691413|3|12| 1|          15
                        | 0|0|1| 54.48| 5|bid|insert|STK.SMART.AIG.
Nov 27 15:58:54 centos-4 : 1838|57534|3128691431|3|12| 1|          15
                        | 1|0|1| 54.46| 39|bid|insert|STK.SMART.AIG.
Nov 27 15:58:54 centos-4 : 1838|57534|3128691447|3|12| 1|          15
                        | 2|2|1| 54.25| 3|bid|delete|STK.SMART.AIG.
Nov 27 15:58:54 centos-4 : 1838|57534|3128691463|3|12| 1|          15
```

```
Nov 27 15:58:54 centos-4 : 1838|57534|3128691478|3|12| 1| 15
    | 2|0|1| 54.25| 3|bid|insert|STK.SMART.AIG.
    | 0|0|0| 54.49| 10|ask|insert|STK.SMART.AIG.
```

...

TBD: del is not working 070205; indeed, the command is echoed as an add

```
book del 15 3;
```

appears in the log as this error message:

```
Nov 27 15:59:03 centos-4 : 1838|57543|3138246745
    |3| 4| 2| 15| 310|Can'tfind the subscribed market
    depth with tickerId:4|
```

3.4.3 See related

- arg 1 is the Operand, one of: add, del
- arg 2 is the security's ContractID lookup
- arg 3 is the Book Depth lookup, as drawn from the index by UID into DEPTHLIMIT; the uid's there have been structured to match the number of rows ('lines') deep to look – that is uid 1 refers 1 row down, uid 2 refers 2 rows down, and so forth.

3.4.4 Peers

There are three other 'peer' subscriptions: `past` (at: [3.19](#)), and `scan` (at: [3.23](#)), and `tick` (at: [3.24](#))

FIXME - continue expansion of book like past example

3.4.5 Listing of: help book

Refreshed from: shim-071228

3.5 cash - FIXME

FIXME - not yet implemented

3.5.1 Description

3.5.2 Usage

Minimal usage:

produces in the logfile:

3.5.3 Peers

It has a synonym called `exercise`; see `exercise` (at: [3.8](#)) which is also used. `cash` is preferred in `help` system documentation matters.

3.6 dbms - describe the dbms to use

3.6.1 Description

Describe the dbms to use

3.6.2 Usage

Minimal usage:

This example is manually entered, after the init command line option was specified;

```
[herrold@centos-4 shim_071109]$ ./shim --data init
```

```
...
```

Enter the dbms connect parameters via the dbms command, using the format:

```
dbms DbmsName DbmsHost TableSet UserName Password;
```

```
dbms mysql xps400.first.lan rph_testing rph_shim 0;
```

```
Ok
```

Enter the upstream connect values via the feed command, using the format:

```
feed FeedName FeedHost FeedPort;
```

```
feed tws xeon.first.lan 7496;
```

```
Ok
```

The trading shim has finished program initialization, including the construction of successful connections to the database and IB tws.

```
quit;
```

```
[herrold@centos-4 shim_071109]$
```

dbms produces no output in the logfile

3.6.3 Peers

There is one other 'peer' command: `feed` (at [3.9](#)), which is similar in that it permits runtime description of the upstream TWS data feed to and from IB, to which the shim is to connect.

Also, the discussion of the `.shimrc` file (at [5.4.1](#)) provides more examples.

3.6.4 Listing of: help link

Refreshed from: shim-071228

3.7 exec - get execution log FIXME

This form of the comand in obsolete, and pending re-write. (see: the note at the start of Cp. 3).

3.7.1 Description

get execution log

FIXME - not yet implemented

3.7.2 Usage

Minimal usage:

TBD - add a scrape

produces in the logfile:

TBD - add a scrape

3.7.3 See related

TBD: arg 1 value – what is it?

3.8 exercise - FIXME

FIXME - not yet implemented

3.8.1 Description

3.8.2 Usage

Minimal usage:

produces in the logfile:

3.8.3 Peers

It has a synonym called `cash`; see `cash`(at: [3.5](#)) which is also used. `cash` is preferred in help system documentation matters.

3.9 feed - describe the upstream TWS market data feed parameters

3.9.1 Usage

Minimal usage:

This example is manually entered, after the init command line option was specified;

```
[herrold@centos-4 shim_071109]$ ./shim --data init
...
```

```
Enter the dbms connect parameters via the dbms command, using the format:
dbms DbmsName DbmsHost TableSet UserName Password;
dbms mysql xps400.first.lan rph_testing rph_shim 0;
Ok
```

```
Enter the upstream connect values via the feed command, using the format:
feed FeedName FeedHost FeedPort;
feed tws xeon.first.lan 7496;
Ok
```

The trading shim has finished program initialization, including the construction of successful connections to the database and IB tws.

```
quit;
[herrold@centos-4 shim_071109]$
```

feed produces no output in the logfile

3.9.2 Peers

There is one other 'peer' command dbms (at: [3.6](#)) which is similar in that it permits runtime description of the database to which the shim is to connect.

Also, the discussion of the .shimrc file (at [5.4.1](#)) provides more examples.

3.9.3 Antecedents

The feed command is a successor to the now obsolete link command.

3.9.4 Listing of: help link

Refreshed from: shim-071228

3.10 help - command verb help

3.10.1 Description

Display `help` information for a command verb

The help system is the front line of documentation, but carries lots to write and to keep updated. As it varies over time and is the ‘leading edge’ of what it documented, we commend to reader the command line generated

Listing of: subsection after the narrative for a given command. Not all commands are presently covered by the help system.

Please see the next page for an example of a **Listing of:** subsection, in this example of the `help help;` command;

3.10.2 Usage

Minimal usage:

```
help help;  
quit;
```

produces no output to `stdout`; the result is directed to `stderr`.

It leaves the following in the logfile:

```
Nov 27 16:19:37 centos-4 : 3033|58777|1085189015|1| 1| 0|help help;|  
Nov 27 16:21:18 centos-4 : 3033|58878|1186354288|1| 7| 0|quit;|
```

3.10.3 Listing of: help help

Refreshed from: shim-071228

3.11 history - ask history query

3.11.1 Description

Ask a history query

3.11.2 Peers

It has a synonym called `past`; see `past` (at: [3.19](#)) which is also used. `past` is preferred in `help` system documentation matters.

3.12 info - get contract info

This command returns a wealth of information about trading options available on a given Contract ID.

3.12.1 Description

3.12.2 Usage

Minimal usage:

```
select info 144 new;
```

produces in the logfile:

```
15222|60974| 875663793|1|15| 0|info 15 new;|
15222|60974| 875683464|2| 9| 3|1|15|new|
15222|60974| 875746177|3|10| 2|AIG|STK||0.0||SMART|USD|AIG|
    AIG|AIG|4301|0.01||ADJUST,ALERT,ALGO,AON,AVGCOST,
    BASKET,COND,CONDORDER,DAY,DEACTEOD,DIS,GAT,GTC,GTD,
    GTT,HID,ICE,IOC,LIT,LMT,LOC,MIT,MKT,MOC,MTL,NONALGO,
    OCA,OPG,OPGREROUT,REL,RTH,SCALE,STP,STPLMT,SWEEP,
    TIMEPRIO,TRAIL,TRAILLMT,SMART,ARCA,CBSX,CHX,DRCTEDGE,
    EDGEA,IBSX,ISE,ISLAND,LAVA,MIBSX,NYSE,PHLX,TRACKECN,VWAP|1|
```

TBD: also version errors 080512

3.12.3 Peers

There is one other 'peer' command for obtaining contract details in bulk: wild (at: [3.29](#)).

3.12.4 Listing of: help info

Refreshed from: shim-071228

3.13 list - list subscriptions

3.13.1 Description

list the presently subscriptions

FIXME - clean up example – is this correct?

3.13.2 Usage

Minimal usage:

```
tick add 177 1;
list;
quit;
```

FIXME ==- no example known
produces in the logfile:

```
Nov 27 17:36:17 centos-4 : 4267|63377| 15046452|1| 4| 0|list;|
Nov 27 17:36:17 centos-4 : 4267|63377| 15046859|2|14| 1|5|
```

FIXME: seems broken; has no appearant effect
It also produces stderr content:

```
Sub: tick 15 STK SMART AIG
Sub: tick 26 STK SMART AXP
Sub: tick 38 STK SMART CAT
Sub: tick 82 STK SMART HON
Sub: tick 83 STK SMART HPQ
...
Sub: tick 163 STK SMART WFMI
Sub: tick 166 STK SMART WYNN
```

3.13.3 Listing of: help list

Refreshed from: shim-071228

3.14 load - Read, or re-read SubRequest table

3.14.1 Description

Read, or re-read the SUBREQUEST table

3.14.2 Usage

Minimal usage:

```
tick add 15 1;
load;
quit;
```

produces in the logfile:

```
15533|63360| 199264964|1|23| 0|load;|
15533|63360| 199292929|1|17| 0|tick add 15 1;|
15533|63360| 199292952|1|17| 0|tick add 26 1;|
15533|63360| 199292972|1|17| 0|tick add 38 1;|
15533|63360| 199292996|1|17| 0|tick add 82 1;|
15533|63360| 199293014|1|17| 0|tick add 83 1;|
...
15533|63360| 199293257|1|17| 0|tick add 163 1;|
15533|63360| 199293276|1|17| 0|tick add 166 1;|
15533|63360| 199312937|2| 1| 5|          15|AIG|STK||0.00||1|SMART||USD||
15533|63360| 199333943|2| 1| 5|          26|AXP|STK||0.00||1|SMART||USD||
15533|63360| 199354856|2| 1| 5|          38|CAT|STK||0.00||1|SMART||USD||
15533|63360| 199374874|2| 1| 5|          82|HON|STK||0.00||1|SMART||USD||
15533|63360| 199380068|3| 1| 5|          15|1|    56.1|    4| |
      price.outcry.bid. |STK.SMART.AIG.
15533|63360| 199380085|3| 1| 5|          15|2|    56.59|    3| |
      price.outcry.ask. |STK.SMART.AIG.
15533|63360| 199380099|3| 1| 5|          15|4|    56.48|    1| |
      price.summary.last. |STK.SMART.AIG.
...
```

The first part is an enumeration of each Contract ID, as it has a tick request added, and then data, is tick data for the symbols selected by the present state of the SUBREQUEST table

3.14.3 Peers

There is one other somewhat similar 'peer' command: `read` (at: [3.22](#)), which causes a full re-read of the database.

TBD: add a usage example show how to populate the SUBREQUEST table

3.14.4 Antecedents

The `load` command is a successor to the now obsolete `bulk` command.

3.14.5 Listing of: help load

Refreshed from: shim-071228

3.15 news - control bulletins

3.15.1 Description

control bulletins

```
select news Adj;
```

where:

- Adj : one of: all, (?? FIXME - are there more)

3.15.2 Usage

Minimal usage:

```
select news all;  
quit;
```

or

```
cancel news all;
```

(From bin/includes)
produces in the logfile:

```
21425|46122| 11957262|1| 5| 0|select news all;|  
21425|46122| 11957288|2|12| 1|all|  
21425|46130| 20060871|1| 5| 0|cancel news;|  
21425|46130| 20060887|2|13| 1|  
21425|46138| 28052704|1| 4| 0|quit;|
```

3.15.3 Limitation

Caveat: We believe, from observation, that the `news` command is only enabled for real live, production accounts, and not for the `demo` or a `paper` account. We have filed a TAC ticket regarding this, and will update this section once an answer is received.

Update: TAC has closed the bug, indicating they will not address the matter.

3.15.4 See related

3.15.5 Listing of: help news

Refreshed from: shim-071228

3.16 next - ping the TWS

3.16.1 Description

3.16.2 Usage

Check connectivity between the shim and the tws

Minimal usage:

```
select next;  
quit;
```

produces in the logfile:

Example from bin/includes:

```
21388|45964| 470144000|1| 2| 0|select next;|  
21388|45964| 470144019|2| 8| 1|  
21388|45964| 470165832|3| 9| 1|1|  
21388|46015| 520941920|1| 4| 0|quit;|
```

3.16.3 Peers

There is one other 'peer' command for programatically checking connectivity: `ping` (at: [3.20](#)), which when answered, indicates the presence of a live connection between the client and TWS.

3.16.4 Listing of: help next

Refreshed from: shim-071228

3.17 open - check open orders

3.17.1 Description

check details for open orders

Note: Per my notes, it is not functioning presently (071112) as it did last February; it formerly returned, when sent, a fresh enumeration of **open** (working) orders; it now seems to act, like **news** as a general session state configuration option.

FIXME: to test and confirm open option

Note: The TWS configuration option:

```
Configure | Global Configuration |
          API | General | Fire openorder on status' change
```

must be enabled (checked), for **open** to work properly.

3.17.2 Usage

Minimal usage:

```
[existing order on]
```

```
...
```

```
open;
```

```
...
```

– no example in bin/includes 070205

Formerly (February 2007) produced in the logfile:

```
...
```

```
Feb 6 11:20:03 centos-4 : shim|risk|0.28| 6054|40803| 3419794|
    3| 3| 5|1|Submitted|0|2|0.0|1473815805|0|0.0|6055|
```

```
Feb 6 11:20:04 centos-4 : shim|risk|0.28| 6054|40804| 4000872|
    2|10| 0|open;|
```

```
Feb 6 11:20:04 centos-4 logger: 6051 open orders info
```

```
Feb 6 11:20:04 centos-4 : shim|risk|0.28| 6054|40804| 4000894|
    3| 5| 1|
```

```
Feb 6 11:20:04 centos-4 : shim|risk|0.28| 6054|40804| 4022761|
    3| 5| 8|1|AIG|STK||0.0|?|SMART|USD|AIG|BUY|2|LMT|68.0|0.0|
    GTC||DU10126|0|0||6055|1473815805|1|0|0||
    1473815805.0/DU10126/100|||||||0||0|||||||1|0|0|0||3|0|0||0|1|
```

```
Feb 6 11:20:04 centos-4 : shim|risk|0.28| 6054|40804| 4023432|
    3| 3| 5|1|Submitted|0|2|0.0|1473815805|0|0.0|6055|
```

```
...
```

3.17.3 Listing of: help open

Refreshed from: shim-071228

3.18 order - manage a LINEITEM

3.18.1 Description

Manage a LINEITEM

The `order` command, due to the number of TWS API parameters, is more complicated. Much of the complexity is hidden in the table: LINEITEM row referred to by UID, but modifiable parameters must be provided on the command line.

```
wire(Oid,Type,Op,Q,P,Aux,T);
```

where:

- `Oid`: the LINEITEM id, a database uid attribute value of LINEITEM
- `Type`: an order type, e.g., MKT, LMT, STP, or TRAIL
- `Op`: one of: Create, Submit, Modify, Cancel
- `Q`: the quantity
- `P`: the limit price
- `Aux`: the auxiliary price
- `T`: the timeout (just a dummy for now, not yet used)

TBD: insert a: describe LineItem and comment on the fields

3.18.2 Usage

Minimal usage:

```
order(1,MKT,Create,100,0.0,0.0,0);
```

or

```
order(1,MKT,Submit,100,0.0,0.0,0);
```

produces in the logfile:

```
FIXME RPH to supplement
```

TBD: doco OCA

TBD: doco bracket

3.18.3 Peers

It has a synonym called `wire`; see `wire` (at: [3.30](#)) which is also used, `wireis` preferred in `help` system documentation matters.

3.19 past - ask history query

This form of the comand is obsolete, and pending re-write. (see: the note at the start of Cp. 3).

3.19.1 Description

ask history query

```
past Op Cid I timeSpec;
```

where:

- Op: one of: add, del
- Cid: the contract id

Note: we discuss looking up a Contract ID in great detail later in this work at: [8.1.3 Looking up a CONTRACT.UID with join](#).

- I: the configuration id, into PASTFILTER, which we will describe in more detail below.
- timeSpec: a specification of the ending time of the query, in one of the following forms:
 - now
where this is taken as the present date and time
 - Ymd_T()
where the argument is of the form: 20070921 15:16:50
Note: that there are two blank spaces seperating the YYYYMMDD and the HH:MM:SS
 - Epoch()
where the argument is in seconds since Unix Epoch

QUERY: Is epoch different from Epoch; a prior testing script example used all lower case.

Note: We observe that if one sends along an ill-formed ending time Ymd_T value, one gets an error message of the following form from the TWS:

```

15:26:54:200 JTS-EServerSocket-655: Error: can't parse
      long string - java.lang.StringIndexOutOfBoundsException:
      String index out of range: 8
15:26:54:201 JTS-EServerSocket-655: [19163:23:35:1:0:0:0:ERR] -
      'rb' : cause - Historical data query end date/time
      string [200709 13:00:00] is invalid. Format is
      'YYYYMMDD{SPACE}hh:mm:ss[{{SPACE}TMZ}']'.
15:26:54:201 JTS-EServerSocket-655: Anticipated error
      jextend.d: Historical data query end date/time string
      [200709 13:00:00] is invalid. Format is
      'YYYYMMDD{SPACE}hh:mm:ss[{{SPACE}TMZ}']'.

```

Now to expand on l, the configuration id, in more detail:

l: the configuration id, points to a row in PASTFILTER by its uid index. The row entry itself is either yet a further pointer off into a sub-table value, or an ultimate fundamental natural unit, That table is described thus:

```

mysql> describe PastFilter ;
+-----+-----+-----+-----+-----+-----+
| Field | Type                | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| uid   | int(10) unsigned   | NO   | PRI | NULL    | auto_increment |
| tid   | int(10) unsigned   | NO   | MUL |         |                 |
| period | int(10) unsigned  | YES  | MUL | NULL    |                 |
| reps  | int(10) unsigned   | NO   |     | 0       |                 |
| duration | int(10) unsigned | NO   |     |         |                 |
| script | char(64)           | NO   |     |         |                 |
+-----+-----+-----+-----+-----+-----+

```

which control:

- PASTFILTER.TID: a pointer through the HISTORYTAG.UID, and then either enumerating a value, or that row itself in turn pointing off to a BARSIZE.UID selected by the value in the HISTORYTAG.BAR field for that row:

```
mysql> describe HistoryTag;
+-----+-----+-----+-----+
| Field  | Type                                | Null | ... |
+-----+-----+-----+-----+
| uid    | int(10) unsigned                   | NO   |     |
| rth_only | tinyint(1)                         | NO   |     |
| format  | enum('ymdt','epoch')               | NO   |     |
| what    | enum('TRADES','MIDPOINT','BID','ASK','BID/ASK') | NO   |     |
| bar    | int(10) unsigned                   | NO   |     |
+-----+-----+-----+-----+
```

which is a truncated form of that DESCRIBE result.

In similar fashion, we wrap the enum MySQL TYPE specification for BARSIZE.

```
mysql> describe BarSize ;
-----+-----+-----+-----+
| Field | Type                                | Null | Key | Default | Extra |
-----+-----+-----+-----+
| uid   | int(10) unsigned                   | NO   | PRI |          |       |
| type  | enum('s01','s05','s15','s30',
      'm01','m02','m05','m15','m30',
      'h01','d01')                   | NO   |     |          |       |
| secs  | int(10) unsigned                   | NO   |     |          |       |
+-----+-----+-----+-----+
```

FIXME - RPH - 071126 – as this is now a permuted table, re-write
– showing all RTH_ONLY (binary), WHAT (TRADES, MIDPOINT, BID, ASK, BID/ASK), FORMAT (ymdt, epoch), and BAR (widths – 1 to 11, from BARWIDTH). This is 220 [= 2x5x2x11] rows.

FIXME – rework this section

For shim tarball versions released after September 2007, we decided to fully pre-populate the HISTORYTAG table with a ‘cross-product’ (that is, a full expansion) of all valid combinations of the two HISTORYTAG.RTH_ONLY values, by the two HISTORYTAG.FORMAT values, by the five HISTORYTAG.WHAT enumeration of values, by the 11 rows in BARSIZE, to yield 220 [= 2x2x5x11] rows.

```
mysql> select count(*) from HistoryTag;
+-----+
| count(*) |
+-----+
|         220 |
+-----+
```

As an example, to seek ‘one second’ history, returned in the ‘ymdt’ format, we can see the available subset of candidate rows in HISTORYTAG meeting that criteria thus:

```
mysql> select * from HistoryTag left
      join BarSize on HistoryTag.bar = BarSize.uid
      where BarSize.type like 's01' and
            HistoryTag.format = 'ymdt' ;
```

uid	rth_only	format	what	bar	uid	type	secs
HistoryTag				(added line)			
				BarSize			
1	0	ymdt	TRADES	1	1	s01	1
12	0	ymdt	MIDPOINT	1	1	s01	1
23	0	ymdt	BID	1	1	s01	1
34	0	ymdt	ASK	1	1	s01	1
45	0	ymdt	BID/ASK	1	1	s01	1
111	1	ymdt	TRADES	1	1	s01	1
122	1	ymdt	MIDPOINT	1	1	s01	1
133	1	ymdt	BID	1	1	s01	1
144	1	ymdt	ASK	1	1	s01	1
155	1	ymdt	BID/ASK	1	1	s01	1

10 rows in set (0.00 sec)

Note: This cross-product obsoletes a discussion elsewhere in this reference to the mechanism by which one would add new PASTFILTER detail specifications.

FIXME: to move that other text to an archival section of instructive obsoleted matter, passed by in developmental changes.

So if we wanted restrict this further to a simple RTH TRADES in ‘ymdt’ reply message format, with ‘one second’ value for a half-hour duration, we would use a PASTFILTER.UID value found as follows:

```
mysql> select PastFilter.uid, PastFilter.duration,
           HistoryTag.rth_only, HistoryTag.format,
           HistoryTag.what, BarSize.type from PastFilter
           left join HistoryTag on PastFilter.tid = HistoryTag.uid
           left join BarSize on HistoryTag.bar = BarSize.uid
           where PastFilter.duration = '1800' and
                  BarSize.type like 's01' and
                  HistoryTag.format = 'ymdt' ;
```

```
+-----+-----+-----+-----+-----+-----+
| uid | duration | rth_only | format | what  | type |
+-----+-----+-----+-----+-----+-----+
|  21 |    1800 |         1 | ymdt  | TRADES | s01  |
+-----+-----+-----+-----+-----+-----+
```

- PASTFILTER.PERIOD: the delay, in seconds, before any repeated `past` History query is re-performed; see also, the next field: PASTFILTER.REPS
- PASTFILTER.REPS: the quantity, i.e., number of times (periods), which the query is repeated; if zero, a single time with no periodic repetition. This defaults to zero.
QUERY: does this then imply that a value of 1 has two instances?
- PASTFILTER.DURATION: the quantity, i.e., number of samples (always in seconds) for the total maximum sample to look back.

TBD: show a RTH which shortens the lookback span, and how it is handled

- PASTFILTER.SCRIPT: what, if any, script to `exec` after the History data is retrieved.

Note: We can make no representation as to whether any such script is run before, during, or after the MySQL `INSERT` of the result of that `past` command query; in part this is because the timing one would observe is not expressly knowable in all instances, absent additional design, coding, and debugging of fairly complex result code checking, retry, failure handling, and ultimately performance limiting case checking into the shim. We choose not to proceed down that path at present.

Instead, the script in question is passed an argument, containing a well-formed and predictably named file name [e.g., `STK.SMART.WYNN.hql`], with the query results, and upon which it may rely as to completeness according to the semantics of the underlying operating system. The

‘tuple’ consists of an expansion of the CONTRACT.UID values threaded back with a LEFT JOIN for: SECTYPE.TYPE, a dot, EXCHANGE.NAME, a dot, and SYMBOL.NAME.

```
mysql> select Contract.uid, SecType.type,
           Exchange.name, Symbol.name from Contract
           left join Symbol   on Contract.sid   = Symbol.uid
           left join SecType  on Symbol.tid    = SecType.uid
           left join Exchange on Contract.route = Exchange.uid
           where Contract.uid = '166' ;
```

uid	type	name	name
166	STK	SMART	WYNN

Note: We do not presently include a mechanism for a script to disambiguate a Future or Option front month from a prior (expired), or longer date expiration, nor to determine a Put from a Call, nor a strike price for an Option. This is on the assumption here, as with the expansions in file output, syslog entries, and so forth, that the downstream client already knows what it is asking for, and can infer the rest from the response context.

TBD: describe how called this exec relates to man 3 exec

TBD: describe how the argument file argv is handed to the child script

3.19.2 Quick lookup of PastFilter configuration id's

There are a limited number of useful configuration id's, I, set out in the PASTFILTER table, which we split in the following database query, to first group by a set limited to RTH (“regular trading hours”) only, and then to the full calendar day of trading hours; the second split, not indicated by the blank line, is into ascending number of seconds covered by a query.

```
mysql> select PastFilter.uid, BarSize.type,
      BarSize.secs as secs, HistoryTag.what,
      HistoryTag.rth_only as rth, PastFilter.duration,
      HistoryTag.format
from PastFilter, HistoryTag, BarSize
where PastFilter.tid = HistoryTag.uid and
      HistoryTag.bar = BarSize.uid and
      HistoryTag.format != 'epoch'
order by rth, secs;
```

uid	type	secs	what	rth	duration	format
12	s01	1	TRADES	0	1800	ymdt
13	s05	5	TRADES	0	9000	ymdt
14	s15	15	TRADES	0	27000	ymdt
15	s30	30	TRADES	0	54000	ymdt
16	m01	60	TRADES	0	1	ymdt
44	m01	60	TRADES	0	30600	ymdt
17	m02	120	TRADES	0	2	ymdt
18	m05	300	TRADES	0	6	ymdt
19	m15	900	TRADES	0	2	ymdt
20	m30	1800	TRADES	0	4	ymdt
21	h01	3600	TRADES	0	10	ymdt
22	d01	86400	TRADES	0	52	ymdt

42	s01	1	TRADES	1	30	ymdt
1	s01	1	TRADES	1	1800	ymdt
2	s05	5	TRADES	1	9000	ymdt
47	s05	5	TRADES	1	30	ymdt
48	s05	5	TRADES	1	30	ymdt
3	s15	15	TRADES	1	27000	ymdt
4	s30	30	TRADES	1	54000	ymdt
23	m01	60	TRADES	1	60	ymdt
24	m01	60	TRADES	1	120	ymdt
25	m01	60	TRADES	1	240	ymdt
26	m01	60	TRADES	1	300	ymdt
27	m01	60	TRADES	1	450	ymdt
28	m01	60	TRADES	1	600	ymdt
29	m01	60	TRADES	1	720	ymdt
30	m01	60	TRADES	1	900	ymdt
31	m01	60	TRADES	1	1800	ymdt
43	m01	60	TRADES	1	23400	ymdt
49	m01	60	TRADES	1	30	ymdt
5	m01	60	TRADES	1	1	ymdt
6	m02	120	TRADES	1	2	ymdt
7	m05	300	TRADES	1	6	ymdt
8	m15	900	TRADES	1	2	ymdt
9	m30	1800	TRADES	1	4	ymdt
45	h01	3600	TRADES	1	1	ymdt
10	h01	3600	TRADES	1	10	ymdt
46	d01	86400	TRADES	1	1	ymdt
11	d01	86400	TRADES	1	52	ymdt

Note: Epoch based past queries are not supported as of 19 May 2008, and so we omit them here in the selection of HISTORYTAG.FORMAT values. Queries with the now timeSpec are able to use ymdt formatted values.

3.19.3 Usage

Minimal usage:

```
select past 177 1 Ymd_T(20070921 15:16:50);
```

which asks for a history dataset, and inserts the return into the shim's database.

TBD: add a section regarding removing content which needs to persist to another database – draft written in dump-data.php

produces in the logfile:

```
Feb 6 11:29:38 centos-4 : shim|data|0.28| 6225|41378| 6600225|
3|17| 2|177|30|
```

Note that history detail lines do NOT carry the shim timestamp.

```
Feb  6 11:29:38 centos-4 : shim|data|0.28| 6225|
    0| 1| 1|0|1|1|20070206 11:29:08|12681.0|12681.0|12681.0|
    12681.0|    26|12681.0|false|FUT.SMART.YM.
Feb  6 11:29:38 centos-4 : shim|data|0.28| 6225|
    0| 1| 1|0|1|1|20070206 11:29:09|12681.0|12681.0|12681.0|
    12681.0|    0|12681.0|false|FUT.SMART.YM.
...
Feb  6 11:29:38 centos-4 : shim|data|0.28| 6225|
    0| 1| 1|0|1|1|20070206 11:29:37|12683.0|
    12683.0|12683.0|12683.0|    0|12683.0|false|FUT.SMART.YM.
Feb  6 11:29:39 centos-4 : shim|data|0.28| 6225|41378|    6638595|
    4|100| 5|# |4|100|5|event: history insert|(177, 1, 2007
    0206 11:29:08 -- 20070206 11:29:37)|
Feb  6 11:29:39 centos-4 : shim|data|0.28| 6225|41379|    7543195|
    2| 7| 0|quit;|
```

The last line before the `quit;` notice is an advice from the shim, that a MySQL `INSERT` has been initiated with the database server containing the retrieved data, in an appropriate form which is mindful of `TBD: name the original v. smoothed forms`.

3.19.4 Extended example

Specific Usages:

This example has appeared in the sample scripts in the recent past.

Note: The following example does MySQL `SELECT` operations against table contents from a pre-October 2007 version of the shim. The particular row contents which a reader may observe will almost certainly contain different `UID` values, than existed at that point in time for tables: `PASTFILTER`, `BARSIZE`, and `HISTORYTAG`. This is a byproduct of the fact that in a SQL database, the sequencing of rows is *not* guaranteed absent a `GROUP BY` or `ORDER BY` clause, through which an ordering is imposed.

```
select past 180 11 now;
```

which we can decode to give a ‘tuple’ for the Contract in question, using its `CONTRACT.UID: 180`

```
mysql> select Contract.uid, Contract.sid, SecType.type, Exchange.name,
->          Symbol.name, FutDetail.expiry from Contract
->    left join Symbol    on Contract.sid    = Symbol.uid
->    left join SecType   on Symbol.tid     = SecType.uid
->    left join Exchange  on Contract.route = Exchange.uid
->    left join FutDetail on Contract.tag   = FutDetail.uid
->          where Contract.uid = '180' ;
```

uid	sid	type	name	name	expiry
180	5494	FUT	SMART	YM	200712

As noted, the third argument is a tag, pointing into the PASTFILTER table. This table in turn has sub-tables feeding it, so that value needs further decoding for easier human use:

```
mysql> select * from PastFilter where PastFilter.uid = '11' ;
```

uid	tid	period	reps	duration	script
11	2	3	0	30	hql2ps

and in doing the full MySQL LEFT JOIN based expansion, we end up with a query like this:

```
select PastFilter.uid, HistoryTag.what, HistoryTag.format,
       HistoryTag.rth_only, BarSize.type, PastFilter.script
       from PastFilter
       left join HistoryTag on PastFilter.tid = HistoryTag.uid
       left join BarSize   on HistoryTag.bar = BarSize.uid
       where PastFilter.uid = '11' ;
```

which yields on our reference database:

```
mysql> select PastFilter.uid, HistoryTag.what, HistoryTag.format,
->          HistoryTag.rth_only, BarSize.type, PastFilter.script
->          from PastFilter
->          left join HistoryTag on PastFilter.tid = HistoryTag.uid
->          left join BarSize   on HistoryTag.bar = BarSize.uid
->          where PastFilter.uid = '11' ;
```

```
+-----+-----+-----+-----+-----+-----+
| uid | what  | format | rth_only | type | script |
+-----+-----+-----+-----+-----+-----+
|  11 | TRADES | ymdt   |          | 1 | s05 | hql2ps |
+-----+-----+-----+-----+-----+-----+
```

So for that complete example we are going to see contract:

FUT — SMART — YM — 200712

with Trades at 5 second intervals with data from regular trading hours, which then also runs the script: `hql2ps` at each history reply message series end.

TBD: RPH: results – perhaps this next section should be back in the later narrative.

As an example on how a end user might extend the History data to be retrieved, assume that we are interested in harvesting HISTORY detail for a couple of timeframes not presently in the PASTFILTER table using a shim tarball from before October 2007. We will again look at FUT.SMART.YM.200712 as follows:

Example	Timeframe	Covered Hours
1	1 second	rth only
2	1 minute	rth only
3	1 minute	wrap rth with pre and post

But, with a shim tarball pre-dating Octoer 2007, we needed to add some relevant new query time frames mentioned in that table. We demonstrate how we would do this using the MySQL INCLUDE fragment:

```
[herrold@centos-4 shim]$ cat rph_hx.sql
--      rph_hx.sql
--
--      add some new history query PastFilter intervals
--
--      table addition needs at least user: 'code' rights
--
--      non rth queries by minute
insert into HistoryTag (bar, what, format, rth_only)
select BarSize.uid, 'TRADES', 'ymdt', '0' from BarSize
```

```

        where BarSize.type = 'm01' ;

--      hour hour of YM by second
--      1800 seconds in a half hour
insert into PastFilter (tid, reps, duration)
      select BarSize.uid, '0', '1800' from BarSize
      where BarSize.type = 's01';

--
--      YM rth matching NYSE hours by minute
--      for 0930-1600 (== 390 minutes, 23400 sec)
insert into PastFilter (tid, reps, duration)
      select BarSize.uid, '0', '23400' from BarSize
      where BarSize.type = 'm01' ;

--
--      YM wider (include some non rth) hours by minute
--      for 0830 to 1645 (== 495 minutes, 30600 sec) AND
--      the new BarSize non-RTH entry
--      we add the left join here
insert into PastFilter (tid, reps, duration)
      select HistoryTag.uid, '0', '30600' from HistoryTag
      left join BarSize on HistoryTag.bar = BarSize.uid
      where BarSize.type = 'm01' and
      HistoryTag.what = 'TRADES' and
      HistoryTag.format = 'ymdt' and
      HistoryTag.rth_only = '0' ;
[herrold@centos-4 shim]$

```

The added line or lines (for HISTORYTAG and PASTFILTER, respectively) are shown at the end of each table dump:

```

mysql> select * from HistoryTag ;
+-----+-----+-----+-----+-----+
| uid | bar | what  | format | rth_only |
+-----+-----+-----+-----+-----+
|  1 |  1 | TRADES | ymdt  |         1 |
| ... |
|  5 |  5 | TRADES | ymdt  |         1 |
| ... |
| 11 | 11 | TRADES | ymdt  |         1 |
| 12 |  5 | TRADES | ymdt  |         0 |
+-----+-----+-----+-----+-----+

```

```
mysql> select * from PastFilter ;
+-----+-----+-----+-----+-----+-----+
| uid | tid | period | reps | duration | script |
+-----+-----+-----+-----+-----+-----+
|  1 |  1 |  NULL |   0 |      30 |        |
|   |   |   ... |   |   |   |
|  5 |  5 |  NULL |   0 |     300 |        |
|   |   |   ... |   |   |   |
| 12 |  5 |    7 |   0 |      30 | hql2ps |
| 13 |  1 |  NULL |   0 |     1800 |        |
| 14 |  5 |  NULL |   0 |    23400 |        |
| 15 | 12 |  NULL |   0 |    24300 |        |
+-----+-----+-----+-----+-----+-----+
```

And to ease understanding that MySQL INSERT example, we show the one second, one minute, and other BARWIDTH values ending in 01:

```
mysql> select * from BarSize where BarSize.type like '%01' ;
+-----+-----+-----+
| uid | type | secs |
+-----+-----+-----+
|  1 | s01 |    1 |
|  5 | m01 |   60 |
| 10 | h01 |  3600 |
| 11 | d01 | 86400 |
+-----+-----+-----+
```

so we can build up a query set like this:

```
ping YM one second bar set at EoD for 21 Sep 2007 ending at 16:00:00;
select past 180 13 Ymd_T(20070921 16:00:00);
quit;
```

We see can see the effect of these commands by cleaning out the database from some prior experimentation (As HISTORYBAR is a table only written to by the shim, the shim is indifferent to the starting uid, or any gaps in the detail present in that table.) The MySQL DELETE FROM operator does not reset the uid counter to an initial state of 0. Accordingly our HistoryBar.uid detail line values do not commence at 0 in this example.

```
mysql> delete from HistoryBar;
Query OK, 1800 rows affected (0.07 sec)
```

```
mysql> select * from HistoryBar ;
Empty set (0.00 sec)
```

And then we run the commands to harvest the history of the last half-hour of the ‘regular trading hours’ OHLC data, at one second summarization intervals:

```
select past 180 13 Ymd_T(20070921 16:00:00);
quit;
```

Turning back to examine the database:

```
mysql> select * from HistoryBar limit 2 ;
```

```

+-----+-----+-----+-----+-----+-----+-----+
| uid | cid | bid | time           | open      | high      |
| low | close | vol | wap           | has_gaps |
+-----+-----+-----+-----+-----+-----+
| 1802 | 180 | 1 | 2007-09-21 15:30:00 | 13929.0000 | 13929.0000 |
| 13929.0000 | 13929.0000 | 5 | 13929.0000 | 0 |
| 1803 | 180 | 1 | 2007-09-21 15:30:01 | 13930.0000 | 13931.0000 |
| 13930.0000 | 13931.0000 | 19 | 13930.0000 | 0 |

```

```
mysql> select * from HistoryBar;
```

```

...
| 3600 | 180 | 1 | 2007-09-21 15:59:58 | 13909.0000 | 13909.0000 |
| 13909.0000 | 13909.0000 | 2 | 13909.0000 | 0 |
| 3601 | 180 | 1 | 2007-09-21 15:59:59 | 13908.0000 | 13909.0000 |
| 13907.0000 | 13908.0000 | 39 | 13908.0000 | 0 |

```

```
1800 rows in set (0.03 sec)
```

So we have demonstrated the retrieval of one second data from upstream. Retrieving a full day’s data is not much different

```
ping YM one minute bar RTH set at EoD for 21 Sep 2007 ending at 16:00:00;
select past 180 14 Ymd_T(20070921 16:00:00);
quit;
```

FIXME: pacing misbehaviour noted: The foregoing command sequence, pasted all at once, does not wait for the pending History return to clear. RFE: Can quit; wait for the history retrieval timeout, if there is an active query in flight?

```

Oct  8 14:48:06 centos-4 : 30074|53286| 2551597|1| 2| 0|
    ping YM one minute bar RTH set at EoD for 21 Sep
    2007 ending at 16:00:00;|
Oct  8 14:48:06 centos-4 : 30074|53286| 2551765|1|19| 0|
    past add 180 14 Ymd_T;|
Oct  8 14:48:06 centos-4 : 30074|53286| 2551842|1| 7| 0|
    quit;|
Oct  8 14:48:06 centos-4 : 30074|53286| 2551908|2|20| 3|
    1|180|14|Ymd_T|

```

But grant a one second sleep (in the external client process feeding the shim), and it works:

```

ping YM one minute bar RTH set at EoD for 21 Sep 2007 ending at 16:00:00;
select past 180 14 Ymd_T(20070921 16:00:00);
[ sleep 1 ]
quit;

```

And indeed, when run as a second connection, with a make test running in a second connection panel, spread out in time, and without the quit; being stated until after an arbitrary delay (here, eight minutes, from 15:00:37 to 15:08:40 for process ID: 30696), and then run a second time as well, it works fine:

```

...
Oct  8 15:00:37 centos-4 : 30696|54037| 16140163|3| 1| 1|
    20070921 15:59:00|13911.0|13915.0|13907.0|13908.0|
    496|13911.0|false|FUT.SMART.YM.
Oct  8 15:00:37 centos-4 : 30696|54037| 16651263|4|100| 5|
    # |4|100|5|event: history insert|
    (180, 5, 20070921 09:30:00 -- 20070921 15:59:00)
...
Oct  8 15:08:40 centos-4 : 30696|545| 499671050|1| 7| 0|quit;|

```

And to complete the set of examples, with a retrieval including data from outside of 'regular trading hours', using the new PASTFILTER.UID = 15 query that we added earlier:

```

Oct  8 15:25:15 centos-4 : 31639|55515| 117661378|1|19| 0|
    past add 180 15 Ymd_T;|
Oct  8 15:25:15 centos-4 : 31639|55515| 117681245|2|20| 3|
    3|180|15|Ymd_T|
Oct  8 15:25:15 centos-4 : 31639|55515| 118049738|3|17|

```

```

|          180|510|
Oct  8 15:25:15 centos-4 : 31639|55515| 118038614|3| 1| 1|
    20070921 08:15:00|13917.0|13921.0|13917.0|13921.0
    |    40|13920.0|false|FUT.SMART.YM.
Oct  8 15:25:15 centos-4 : 31639|55515| 118038637|3| 1| 1|
    20070921 08:16:00|13921.0|13923.0|13920.0|13923.0
    |    29|13922.0|false|FUT.SMART.YM.
...
Oct  8 15:25:16 centos-4 : 31639|55515| 118049567|3| 1| 1|
    20070921 16:43:00|13900.0|13900.0|13900.0|13900.0
    |    0|13900.0|false|FUT.SMART.YM.
Oct  8 15:25:16 centos-4 : 31639|55515| 118049587|3| 1| 1|
    20070921 16:44:00|13900.0|13901.0|13900.0|13901.0
    |    4|13901.0|false|FUT.SMART.YM.
Oct  8 15:25:16 centos-4 : 31639|55515| 118847318|4|100| 5|
    # |4|100|5|event: history insert|
    (180, 5, 20070921 08:15:00 -- 20070921 16:44:00)|

```

showing data spanning from 08:15:00 to 16:44:00.

TBD: describe further the `INSERT` mechanism into `HISTORYBAR`, the optional script call, and highlighting the need to harvest anything needed more permanently with our sample `HISTORYBAR` extract script.

3.19.5 Peers

There are three other 'peer' subscriptions: `book` (at: [3.4](#)), `scan` (at: [3.23](#)), and `tick` (at: [3.24](#)).

It has a synonym called `history`; see `history` (at: [3.11](#)) which is also used. `past` is preferred in `help` system documentation matters

3.19.6 Listing of: help past

Refreshed from: shim-071228

3.20 ping - log time, comment through EOL

3.20.1 Description

Pass through a time mark, and comment through EOL; A successful return also verifies a presently live connection from the client through the shim to the upstream TWS.

3.20.2 Usage

Minimal usage:

The commands: `ping` alone is unusual in that it alone take optional comment content through the next semicolon: `;`

```
ping acme;
quit;
```

produces in the logfile:
(example from `bin/includes`)

```
Feb  5 21:29:34 centos-4 : shim|data|0.28| 2476|77374|    5148896|
      2| 2| 0|ping acme;|
Feb  5 21:29:44 centos-4 : shim|data|0.28| 2476|77384|    14649744|
      2| 7| 0|quit;|
```

3.20.3 Peers

There is one other 'peer' command for programatically checking connectivity: `next` (at: [3.16](#)), which when answered, indicates the presence of a live connection between the TWS and IB (and implicitly between the client, the shim, and the TWS).

3.20.4 Listing of: help ping

Refreshed from: shim-071228

3.21 quit - exit the program

3.21.1 Description

That's it for now

3.21.2 Usage

Minimal usage:

```
quit;
```

(From bin/includes)

produces in the logfile (this is the full session, of a minimum possible well formed shim session):

```
Feb  5 18:08:59 centos-4 : shim|data|0.28|29075|65338| 1022377|
      4|100| 5|# |4|100|5|*****|
Feb  5 18:08:59 centos-4 : shim|data|0.28|29075|65338| 1022384|
      4|100| 5|# |4|100|5|version|0.28|070202|
Feb  5 18:08:59 centos-4 : shim|data|0.28|29075|65338| 1022387|
      4|100| 5|# |4|100|5|*****|
Feb  5 18:08:59 centos-4 : shim|data|0.28|29075|65338| 1022483|
      3| 9| 1|1|
Feb  5 18:08:59 centos-4 : shim|data|0.28|29075|65339| 1335811|
      3| 4| 2|      -1|2104|Market data farm connection is OK:usfuture|
Feb  5 18:08:59 centos-4 : shim|data|0.28|29075|65339| 1335841|
      3| 4| 2|      -1|2104|Market data farm connection is OK:usfarm|
Feb  5 18:08:59 centos-4 : shim|data|0.28|29075|65339| 1335871|
      3| 4| 2|      -1|2107|HMDS data farm connection is inactive
      but should be available upon demand.:ushmds2a|
Feb  5 18:09:18 centos-4 : shim|data|0.28|29075|65358| 21039847|
      2| 7| 0|quit;|
```

See also: wait (at: [3.27](#)), wake (at: [3.28](#))

3.21.3 Listing of: help quit

Refreshed from: shim-071228

3.22 read - FIXME

3.22.1 Description

re-read the entire database, adding new rows found.

It is a mandatory precursor to some commands. After a order detail line has been added to the `LINEITEM` table, the `read` command is required, before that newly added line is known to and therefore properly able to be used by the shim, as in an `order` (see: [3.18](#)) command.

3.22.2 Usage

Minimal usage:

produces in the logfile:

3.22.3 Peers

There is one other somewhat similar 'peer' command `load` (at: [3.14](#)), which also causes a re-read of the database.

3.22.4 Antecedents

The `read` command is a successor to the now obsolete `look` command.

3.22.5 Listing of: help read

Refreshed from: shim-071228

3.23 scan - event scanner TBD

FIXME - not yet implemented

3.23.1 Description

event scanner TBD

```
select scan  FIXME;
```

FIXME - not yet implemented

3.23.2 Usage

Minimal usage:

TBD - add a scrape

produces in the logfile:

TBD - add a scrape

TBD - add a scrape

3.23.3 Peers

There are three somewhat similar 'peer' subscriptions: `book` (at: [3.4](#)), `past` (at: [3.19](#)), and `tick` (at: [3.24](#)). Really, the market scanner does not match the syntax of those commands very closely.

3.24 tick - subscribe mk data

3.24.1 Description

subscribe to market data

```
select tick Cid I;
```

where:

- Cid: the contract id, from CONTRACT.UID
- I: the configuration id, from TICKCONFIG.UID

cancel the subscription to market data

```
cancel tick Cid I;
```

Also, the form:

```
cancel tick Cid;
```

works.

3.24.2 Usage

Minimal usage:

```
select tick 142 1;  
quit;
```

Example in bin/includes

```
select tick 142 1;
```

produces in the logfile:

```
21384|45153| 240241062|3| 1| 5|      142|4| 12837.0|      2|  
    |price.summary.last. |FUT.ECBOT.YM.  
21384|45153| 240241076|3| 2| 5|      142|8|                60119|0|size.volume.  
    |FUT.ECBOT.YM.  
21384|45155| 241989519|3| 2| 5|      142|3|                4|0|size.ask.  
    |FUT.ECBOT.YM.  
21384|45156| 243238594|3| 2| 5|      142|3|                5|0|size.ask.  
    |FUT.ECBOT.YM.  
21384|45158| 245244044|3| 2| 5|      142|3|                6|0|size.ask.
```

```

      |FUT.ECBOT.YM.
21384|45159| 246244727|3| 2| 5|      142|3|      5|0|size.ask.
      |FUT.ECBOT.YM.
21384|45159| 246244739|3| 2| 5|      142|5|      1|0|size.last.
      |FUT.ECBOT.YM.
21384|45159| 246244750|3| 2| 5|      142|8|      60120|0|size.volume.
      |FUT.ECBOT.YM.
21384|45160| 247243801|3| 2| 5|      142|3|      7|0|size.ask.
      |FUT.ECBOT.YM.
21384|45161| 248243034|3| 2| 5|      142|0|      7|0|size.bid.
      |FUT.ECBOT.YM.
21384|45163| 249998586|3| 2| 5|      142|0|      8|0|size.bid.
      |FUT.ECBOT.YM.
21384|45164| 250995775|3| 2| 5|      142|0|      7|0|size.bid.
      |FUT.ECBOT.YM.
21384|45165| 251998692|3| 2| 5|      142|3|      5|0|size.ask.
      |FUT.ECBOT.YM.
21384|45167| 254256802|3| 2| 5|      142|0|      8|0|size.bid.
      |FUT.ECBOT.YM.
21384|45169| 256263445|3| 2| 5|      142|0|      7|0|size.bid.
      |FUT.ECBOT.YM.
21384|45170| 257269297|3| 2| 5|      142|0|      8|0|size.bid.
      |FUT.ECBOT.YM.
121384|45171| 258273136|3| 2| 5|      142|0|      9|0|size.bid.
      |FUT.ECBOT.YM.
21384|45171| 258273148|3| 2| 5|      142|3|      4|0|size.ask.
      |FUT.ECBOT.YM.
21384|45172| 258913711|1| 4| 0|quit;|

```

3.24.3 Peers

There are three other 'peer' subscriptions: `book` (at: [3.4](#)), `past` (at: [3.19](#)), and `scan` (at: [3.23](#)).

3.24.4 Future explanation

FIXME: Explain tick argument `TICKCONFIG` feature variation

There is the `I`: Configuration ID tag in the command, which points into the `TICKCONFIG`. We carry it here, to provide full access to all tick request features.

QUERY: What are the variations, and how might they be useful?

```
mysql> describe TickConfig;
```

Field	Type	Null	Key	Default	Extra
uid	int(10) unsigned	NO	PRI	NULL	auto_increment
type	enum('tick','time')	NO		tick	
bar	int(10) unsigned	NO	MUL		
bars	int(10) unsigned	NO		1	

```
mysql> select * from TickConfig ;
```

uid	type	bar	bars
1	tick	5	9
2	time	2	9
3	tick	5	20
4	time	2	20

FIXME - continue expansion of tick like past example

3.24.5 Listing of: help tick

Refreshed from: shim-071228

3.25 transmit - FIXME

3.25.1 Description

FIXME - alias to xmit (at: [3.31](#))

FIXME - add pointer manually

3.25.2 Usage

Minimal usage:

TBD - add a scrape

produces in the logfile:

TBD - add a scrape

3.26 verb - set tws log level

3.26.1 Description

set tws log level

These are several levels available:

```
select verb Level;
```

QUERY: modify is also used in the place of select??

where:

- Level: one of: System, Error, Warning, Information, Detail

3.26.2 Usage

Minimal usage:

```
select verb Detail;
```

```
quit;
```

(From bin/includes)

produces in the logfile:

```
Feb 6 11:44:56 centos-4 : shim|data|0.28| 6447|42296| 1019387|
    4|100| 5|# |4|100|5|*****|
Feb 6 11:44:56 centos-4 : shim|data|0.28| 6447|42296| 1019393|
    4|100| 5|# |4|100|5|version|0.28|070112|
Feb 6 11:44:56 centos-4 : shim|data|0.28| 6447|42296| 1019396|
    4|100| 5|# |4|100|5|*****|
Feb 6 11:44:56 centos-4 : shim|data|0.28| 6447|42296| 1019490|
    3| 9| 1|1|
Feb 6 11:44:56 centos-4 : shim|data|0.28| 6447|42296| 1399452|
    3| 4| 2|      -1|2104|Market data farm connection is OK:usfuture|
Feb 6 11:44:56 centos-4 : shim|data|0.28| 6447|42296| 1399485|
    3| 4| 2|      -1|2104|Market data farm connection is OK:usfarm|
Feb 6 11:44:56 centos-4 : shim|data|0.28| 6447|42296| 1399516|
    3| 4| 2|      -1|2107|HMDS data farm connection is inactive
    but should be available upon demand.:ushmds2a|
Feb 6 11:45:01 centos-4 : shim|data|0.28| 6447|42301| 6100183|
    2| 8| 0|verb Detail;|
Feb 6 11:45:01 centos-4 : shim|data|0.28| 6447|42301| 6100215|
    3|14| 1|5|
Feb 6 11:45:02 centos-4 : shim|data|0.28| 6447|42302| 7318151|
    2| 7| 0|quit;
```

NOTE: as of 080512, this stdout is not present.

3.26.3 Listing of: help verb

Refreshed from: shim-071228

3.27 wait - sleep shim N secs

3.27.1 Description

sleep shim N secs

There is a companion: `wake` which causes an early termination of an active `wait`.

```
wait N;
```

where:

- N: an integer

3.27.2 Usage

Minimal usage:

```
wait 5;
quit;
```

– no example in `bin/includes`
produces in the logfile:

```
Feb  6 12:41:18 centos-4 : shim|data|0.28| 7286|45678|   3696040|
      2| 5| 0|wait 5;|
Feb  6 12:41:21 centos-4 : shim|data|0.28| 7286|45681|   5957462|
      2| 7| 0|quit;|
```

FIXME: more narrative explaining how it will hold up a `quit` but due to the input reader's greedy nature, not much else. Actually this is stale as well, as there are immediate commands to discuss now.

See also: `quit` (at: [3.21](#)), and `wake` (at: [3.28](#))

3.27.3 Listing of: help wait

Refreshed from: shim-071228

3.28 wake - clear pause count

clear a pause count created with `wait`

This is a companion to `wait`, in that it causes an early termination of any running `wait`

3.28.1 Description

clear any pause from the (`wait`) command at once

3.28.2 Usage

Minimal usage:

```
wait 60;
wake;
quit;
```

This fragment produces in the logfile:

```
Feb  5 21:38:17 centos-4 : shim|data|0.28| 2498|77897|   6280957|
      2| 5| 0|wait 60;|
Feb  5 21:38:19 centos-4 : shim|data|0.28| 2498|77899|   8546237|
      2| 6| 0|wake;|
Feb  5 21:38:22 centos-4 : shim|data|0.28| 2498|77902|  11223130|
      2| 7| 0|quit;|
```

That is, the `wake` at line 2 cancels the `wait` at line 1, and permits the `quit` at line 2 proceed earlier than otherwise scheduled.

In the next example, however, and perhaps counterintuitively, the `wake` at line 3 “reached through” the `quit`, to “prematurely” cancel a previously running `wait` at line 1, and then permits the `quit` at line 2 proceed earlier than otherwise scheduled.

```
wait 60;
quit;
wake;
```

This arises from the way the command line queue is read to exhaustion of completed lines, and then given effect so much as possible; the `wait` line 1, the `quit` is in the queue to run in 60 seconds, and then the `wake` at line 3 is encountered, and the running `wake` is ended. The `quit` then immediately follows.

```
Feb  5 21:38:31 centos-4 : shim|data|0.28| 2499|77911| 4076453|
      2| 5| 0|wait 60;|
Feb  5 21:38:34 centos-4 : shim|data|0.28| 2499|77914| 7291713|
      2| 7| 0|quit;|
Feb  5 21:38:38 centos-4 : shim|data|0.28| 2499|77918| 11176296|
      2| 6| 0|wake;|
```

So in reviewing the timestamps, it is clear that each has the same effect of cancelling the `wait`, and then promoting the next following into effect, where this being the `quit` in each case.

See also: `quit` (at: [3.21](#), `wait` (at: [3.27](#))

3.28.3 Listing of: help wake

Refreshed from: shim-071228

3.29 wild - abstract contract

3.29.1 Description

```
select wild FUT ZS all;
```

– companion to `info` (at: [3.12](#)) for wildcard contract information lookup

3.29.2 Usage

Minimal usage:

```
select wild FUT ZS all;
```

produces in the logfile:

```
error as to version 080512
```

produces in the `stderr`:

3.29.3 Peers

There is one other 'peer' command for obtaining details: for a specific contract: `info` (at: [3.12](#)).

3.30 wire - accumulate orders

3.30.1 Description

accumulate orders

3.30.2 Peers

It has a synonym called `order`; see `order` (at: [3.18](#)) which is also used. `wire` is preferred in `help` system documentation matters.

3.30.3 Listing of: help wire

Refreshed from: shim-071228

3.31 xmit - release tws order

3.31.1 Description

release tws order

QUERY: believed obsolete

QUERY – one or many – is this a companion to *wire*?

NOTE: There is a commented out partial of xmit in bin/includes

3.31.2 Usage

Minimal usage:

TBD -- add a scrape

produces in the logfile:

TBD - add a scrape

3.31.3 See related

QUERY: related to *order (wire)* ?

Chapter 4

Parameters, common to the command verbs

4.1 Parameters to the command verbs

4.1.1 Simple parameters

- N: the number of seconds
- Level: one of: System, Error, Warning, Information, Detail
- Cid: the contract id, a database UID attribute value of CONTRACT
- Op: one of: add, del
- I: the configuration id, a database table uid, one of, by command verb:
 - tick: TICKCONFIG
 - book: DEPTHLIMIT
 - past: PASTFILTER

4.1.2 Order (wire) parameters

- Oid: the line item id, a database uid attribute value of LINEITEM
- Type: an order type, e.g., MKT, LMT, STP, or TRAIL
- Op: one of: Create, Submit, Modify, Cancel
- Q: the quantity
- P: the limit price
- Aux: the auxiliary price
- T: the timeout (just a dummy for now, not yet used)

Chapter 5

`'shim -help'` matters

5.1 `-help` - short form help from the program

5.1.1 Description

Earlier through this reference, we placed, per command, the output from running the shim in `-help` mode, for each supported command:

- `acct` at: [3.2.3](#)
- `book` at: [3.4.4](#)
- `info` at: [3.12.3](#)
- `list` at: [3.13.2](#)
- `load` at: [3.14.4](#)
- `news` at: [3.15.4](#)
- `next` at: [3.16.3](#)
- `open` at: [3.17.2](#)
- `past` at: [3.19.5](#)
- `ping` at: [3.20.3](#)
- `quit` at: [3.21.2](#)
- `read` at: [3.22.4](#)
- `tick` at: [3.24.4](#)
- `verb` at: [3.26.2](#)
- `wait` at: [3.27.2](#)
- `wake` at: [3.28.2](#)
- `wire` at: [3.30.2](#)

This leaves the following output available under `-help` mode, yet to print:

- `args` at: [5.1.2](#)
- `cmds` at: [5.1.3](#)
- `help` at: [5.1.1](#)

- link at: [5.1.4](#)
- mode at: [5.2](#)
- opts at: [5.3](#)

which we set forth below.

This section is maintained as a ‘aid to memory’ through mechanical generation, and may lag from time to time the present state of the shim. Of course, in case of uncertainty or conflict, the source code itself is authoritative.

5.1.2 Listing of: help help

Refreshed from: shim-071228

5.1.3 Listing of: help args

Refreshed from: shim-071228

5.1.4 Listing of: help cmds

Refreshed from: shim-071228

5.1.5 Listing of: help link

Refreshed from: shim-071228

5.2 shim Modes

The shim has several modes of operation. Normal day to day operation will use: `data` locks out order logic; or `risk` permits orders. As always, verify through testing your trust level in the shim code before using it with a live account.

1. `data`
2. `risk`

There is a mode to permit starting the shim with no requirement of a TWS connection, nor of a working database. It also permits help subsystem use.

1. `help`

The last two are primarily for developers.

1. `play`
2. `unit`

5.2.1 Listing of: help mode

Refreshed from: shim-071228

5.3 shim Options

The shim has several options which may modify a given modes of operation. Normal day to day operation might use:

1. file
2. cout
3. logd
4. init
5. pane
6. load
7. save
8. fast

We do not describe in great detail them here, but rather leave the description for the following subsection for the present.

5.3.1 Listing of: help opts

Refreshed from: shim-071228

5.4 .shimrc - optional file to describe shim parameters

5.4.1 Usage

Minimal usage:

The shim will colate the connection data needed for the upstream TWS connection, and fo rthe database connection from several sources; A higher priority specification will over-ride an earlier one.

The shim sources ship with default settings which look for a local TWS, and a local database. The `-help` command has full and particular details, and it follows this section.

Basically, the compiled in defaults (which are set out in `src/data.c`) may be over-ridden by:

1. Any `.shimrc` file found at the `$HOME` [or `/`] directory of the UNIX `$USER` running the shim.
2. Any `dbms` or `feed` commands, either from a script file, or as manually entered in repsonse to the prompts for, first `dbms` and then `feed`, but only in such cases as the shim was started with the `init` option.

Note: if the `init` option *is* used, entries for both `dbms` or `feed` are required befor the timeout.

A sample commmand line session appears as follows:

```
[herrold@centos-4 shim_071109]$ ./shim --data init
```

```
...
```

Enter the `dbms` connect parameters via the `dbms` command, using the format:

```
dbms DbmsName DbmsHost TableSet UserName Password;  
dbms mysql xps400.first.lan rph_testing rph_shim 0;
```

```
Ok
```

Enter the upstream connect values via the `feed` command, using the format:

```
feed FeedName FeedHost FeedPort;  
feed tws xeon.first.lan 7496;
```

```
Ok
```

The trading shim has finished program initialization, including the construction of successful connections to the database and IB tws.

```
quit;
```

```
[herrold@centos-4 shim_071109]$
```

Use of an `.shimrc` file, produces no output in the logfile; input manually entered at the command line *is* echoed by the shell, but the shim does not pass them through to the `stdout`, absent the use of the [2](#) `cout` command line option. All the `shim` prompts are to the `stderr`.

We can see where output is being routed by the shell, between the `stdout`, the `stderr`, and echoing of the `stdin` thus. All visible input `stdin` was locally typed, and thus is visible when echoed by the shell (but not from an appearance on the `stdout`).

```
[herrold@centos-4 shim_071210]$ ./shim --data init 2> /dev/null
dbms mysql xps400.first.lan rph_testing rph_shim 0;
feed tws xeon.first.lan 7496;
quit;
[herrold@centos-4 shim_071210]$
```

In the next example, we see that

```
[herrold@centos-4 shim_071210]$ ./shim --data init > /dev/null
Enter the dbms connect parameters via the dbms command, using the format:
dbms DbmsName DbmsHost TableSet UserName Password;
dbms mysql xps400.first.lan rph_testing rph_shim 0;
Ok
```

```
Enter the upstream connect values via the feed command, using the format:
feed FeedName FeedHost FeedPort;
feed tws xeon.first.lan 7496;
Ok
```

The trading shim has finished program initialization, including the construction of successful connections to the database and IB tws.

```
quit;
[herrold@centos-4 shim_071210]$
```

5.4.2 Peers

There are the two ‘peer’ commands to the `init` option: `dbms` (at: [3.6](#)) and, `feed` (at: [3.9](#)) which permit runtime description of the database to which the shim is to connect, or the TWS to which the shim is to connect, respectively.

Chapter 6

Numbering - Commands, Requests, Messages, Comments

The trading shim is a command-line and dbms controlled interface to the socket-based API of Interactive Brokers' Trader Workstation, abbreviated as the IB tws socket api, or simply 'tws'.

– [[trading-shim home page](#)]

6.1 Overview on message numbering

Let's break that down a bit, and consider just the messages of various type, which the shim passes to and receives from the TWS and from its downstream clients; for the present, we put aside the shim's communication with the database.

The shim, under this simplified analytic model, is a tool to accept command line input (Commands). It then consults its view of a database (which we ignore here). The shim emits a well formed series of binary strings across a socket connection to a TWS (Requests). The TWS then replies to the shim with a well formed series of binary strings, again across a socket connection back (Messages). The shim decodes all these messages, and annotates parts of the binary strings, and formats all three message strings, along with adding commentary on some state machine status matters (Comments).

The first three fields of message entries are reflected by small integer numbers in the first three substantive entries of formatted `ShimText` or `logd` output.

6.2 message class, message value and message version

Those first three values represent the message class, the message value, and message version. We are familiar with these from viewing such output:

```
15973|42454| 1033567|4|100| 0|# |4|100|0|*****|
15973|42454| 1033577|4|101| 0|# |4|101|0|0.52|070831|risk|
15973|42454| 1033580|4|100| 0|# |4|100|0|*****|
15973|42454| 1033562|4|102| 0|# |4|102|0|23|15973|39|
      20080114 11:47:33 EST|Connect with: cv 23, id 15973, sv 39|
15973|42454| 1176390|3| 9| 1|1|
15973|42454| 1194530|3| 4| 2|      -1|2104|
      Market data farm connection is OK:u
sfarm|
15973|42454| 1194554|3| 4| 2|      -1|2104|
      Market data farm connection is OK:usfuture|
15973|42454| 1194577|3| 4| 2|      -1|2106|
      HMDS data farm connection is OK:ushmids2a|
15973|42457| 3990801|1|20| 0|past add 181 6 now;|
15973|42464| 10422584|2|20| 3|1|181|6|now|
15973|42464| 10938322|3|17| 3|      181|8|
15973|42464| 10938103|3| 1| 1|20080114 11:40:13|
      12747.0|12751.0|12747.0|12749.0|      89|12749.0|
      false|FUT.SMART.YM.
```

We can use the Unix `cut` and `head` commands to focus on the parts we are interested in discussing:

```
[herrold@centos-5 shim_080114]$ cut -d"|" -f 4-6 ShimText | head -n 12
4|100| 0
4|101| 0
4|100| 0
4|102| 0
3| 9| 1
3| 4| 2
3| 4| 2
3| 4| 2
1|20| 0
2|20| 3
3|17| 3
3| 1| 1
```

6.2.1 message class

Viewed this way, and looking at the leftmost (first) column (message class), we see four Commands (class 4), four Messages (class 3), one Command (class 1), one Request (class 2), and two more Messages (class 3); presently only four class numbers are used by the shim to report the message class.

number	message class
1	Commands
2	Requests
3	Messages
4	Comments

6.2.2 message value

The center (second) field of each line is the message value, which is a uniquely varying series, within each given message class. The message value is assigned by the program that produces it: by the shim for Commands and Comments; by the TWS for Requests and Messages. We will view this in greater detail in the balance of this piece.

6.2.3 message version

The rightmost (third) field is the message version.

In the implementation of the shim, we have not sought to preserve obsolete or outmoded Command, and Comment message value numbers. As such, the shim uses a message version of zero.

IB has a stronger desire to support both prior TWS releases and new additions to its API over time; it therefore provides a message version to each of its message values, which are called Server versions (Messages), and Client versions (Requests).

6.2.4 TWS message value and message version co-ordination

As noted, IB has a trickier task of release engineering with the TWS than the shim project, as there is an installed base of clients which would potentially be broken if IB re-assigned TWS message numbers for Requests or Messages. Broken clients potentially mean a client leaving the IB interfaces, and that is commercially costly. They finessed the issue by simply adding new message value numbers, and occasionally updating the minimum supported message versions, on the part of the Client (the downstream) or the Server (the TWS). Obsolete forms can eventually be discarded by causing the TWS to only accept a minimum Client message version. We note that

when IB has done this from time, it tends to provoke much consternation in the end user community.

This upstream approach on issuing new TWS Request or Message message value assignments, in turn drives the process of extension of the shim as new Request message values emerge, or new Message message values are encountered with a new TWS version or by IB instructing an existing TWS to require a minimum Client message version (both approaches have been used by IB). The portents of such a change coming may be anticipated by study of API release notes from IB, and is often heralded by parser errors on the `stderr` of the shim as well.

When found, the shim is extended by getting a clear definition of the new expected message values (usually from examination of the Java sample client). Then by studying the functions and methods which IB has added, one may then shim patterns to articulate well-formed new Requests, and extract information from new Messages.

6.3 message values in the TWS

The Java sample client is an authoritative source for the mapping between Request numbers, and Message numbers which the shim needs to use in communicating with the TWS; the shim sources are an authoritative source for the mapping between Command numbers, and Comment numbers. As noted above, the combination of these message value number series appears in the logging, to file, to the syslog, and so forth, which the shim does for its downstream users.

6.4 Java sample client

The web documentation of IB is perhaps, naturally, the first secondary source one might turn to, in seeking to understand how the TWS works; This web page

<http://individuals.interactivebrokers.com/php/webhelp/Interoperability/logging.htm>

offers a word of caution, and suggests that:

NOTE: this information, along with the various request/response message versions, can be found in the `EClientSocket` implementation file supplied with the API installation.

The author at IB knew that this resource becomes stale, and incomplete, as the answers will vary with each new API release, Server version, and Client

version. To their credit, it seems that new versions are largely ‘additive’ in expanding features, or new options for existing features. A couple of false starts, and only a few re-definitions are known – history retrieval changes from October 2006 to present; rationalization of the two inverse sense ‘rth’ binary flags.

So, IB itself refers a person wishing to understand its code to its ‘reference implementations.’ This ‘reference’ code is not intentionally obfuscated, is reasonably well versioned, and is freely available (albeit under a ‘non-free’ copyright and license).

6.4.1 Rationale’ for consulting the Java sample client

IB ships several implementations of sample clients. We choose to use the Java one because it seems that the java contained in that client is most likely to be used verbatim in the Java based TWS itself.

Side note: We obtained this URL from the following process (using the Firefox web browser; version 1.5.0.12):

1. Open the web page at:

<http://individuals.interactivebrokers.com/php/webhelp/webhelp.htm>

2. Select the Contents button in the upper left
3. Drag the selection box down to: API
4. Click within that topic: API Logging
5. In the Right Lower panel, at the bottom, it presently states:

NOTE: this information, along with the various request/response message versions, can be found in the EClientSocket implementation file supplied with the API installation.

6.4.2 How to view a permanent page URL on the IB site

IB has started to use non-traditional basic HTML features (javascript, Flash, and such), which assumedly make the site more compelling for humans viewing it; This has the consequence, however, of removing easy to read and reference, permanent URL’s.

Accordingly, we write this side note: How to view the permanent page URL

1. Open the bottom right panel (HTML frame) in a new Window of its own.
2. Right-click to select: View Page Info
3. Expand the box so that the URL can all be seen at the top of the General tab, Address field.
4. Highlight and copy it to the clipboard.

6.4.3 Retrieving the Java sample client

One may use the following method to consult the source files, which we consider authoritative.

1. Obtain the Java sample (standalone) client at:

http://individuals.interactivebrokers.com/en/control/standalone_api.php?os=unix

2. Select and save from the link for the: `twapi_unixmac.jar`

Note: IB does not version these releases. Each is named the same as its predecessor, is at the same URL, and will silently over-write old versions when retrieved; you may wish to implement a system to compare the md5sum of a downloaded version with a prior corpus of downloads, to permit detection and the retention of a newly appearing release.

3. Place the retrieved `twapi_unixmac.jar` in a newly created temporary directory, to avoid inadvertently admixing it with other content.
4. Unpack it:

```
jar xf twapi_unixmac-9.40.jar
```

5. View the version number:

```
$ cat IBJts/API_VersionNum.txt
API_Version=9.40
```

6. Using the Unix `find` command, look for a match on the filename fragment, referenced on the documentation webpage

```
find -name "EClientSocket*"
ls -l ./IBJts/java/com/ib/client/EClientSocket.java
```

7. Print or view the documentation:

```
lpr ./IBJts/java/com/ib/client/EClientSocket.java
less ./IBJts/java/com/ib/client/EClientSocket.java
```

6.5 Numbering in the Java sample client

Several of the message numbers we seek are in the Java sample client. These examples are pulled from the API version 9.40, and will vary over time.

6.5.1 Numbering of Requests in EClientSocket.java

As of API version 9.40, the following Request types are enumerated in EClientSocket.java

```
// outgoing msg id's
private static final int REQ_MKT_DATA = 1;
private static final int CANCEL_MKT_DATA = 2;
private static final int PLACE_ORDER = 3;
private static final int CANCEL_ORDER = 4;
private static final int REQ_OPEN_ORDERS = 5;
private static final int REQ_ACCOUNT_DATA = 6;
private static final int REQ_EXECUTIONS = 7;
private static final int REQ_IDS = 8;
private static final int REQ_CONTRACT_DATA = 9;
private static final int REQ_MKT_DEPTH = 10;
private static final int CANCEL_MKT_DEPTH = 11;
private static final int REQ_NEWS_BULLETINS = 12;
private static final int CANCEL_NEWS_BULLETINS = 13;
private static final int SET_SERVER_LOGLEVEL = 14;
private static final int REQ_AUTO_OPEN_ORDERS = 15;
private static final int REQ_ALL_OPEN_ORDERS = 16;
private static final int REQ_MANAGED_ACCTS = 17;
private static final int REQ_FA = 18;
private static final int REPLACE_FA = 19;
private static final int REQ_HISTORICAL_DATA = 20;
private static final int EXERCISE_OPTIONS = 21;
private static final int REQ_SCANNER_SUBSCRIPTION = 22;
private static final int CANCEL_SCANNER_SUBSCRIPTION = 23;
private static final int REQ_SCANNER_PARAMETERS = 24;
private static final int CANCEL_HISTORICAL_DATA = 25;
```

```

private static final int REQ_CURRENT_TIME = 49;
private static final int REQ_REAL_TIME_BARS = 50;
private static final int CANCEL_REAL_TIME_BARS = 51;

```

Note: we manually inserted a blank line after numbers 25 to point up the non-contiguous nature of the listing, differing from that displayed in the sources in question.

6.5.2 Numbering of Messages in EReader.java

As of API version 9.40, the following Message (also called 'response message') types are enumerated in EReader.java

```

// incoming msg id's
static final int TICK_PRICE           = 1;
static final int TICK_SIZE            = 2;
static final int ORDER_STATUS         = 3;
static final int ERR_MSG               = 4;
static final int OPEN_ORDER           = 5;
static final int ACCT_VALUE           = 6;
static final int PORTFOLIO_VALUE      = 7;
static final int ACCT_UPDATE_TIME     = 8;
static final int NEXT_VALID_ID       = 9;
static final int CONTRACT_DATA        = 10;
static final int EXECUTION_DATA       = 11;
static final int MARKET_DEPTH        = 12;
static final int MARKET_DEPTH_L2     = 13;
static final int NEWS_BULLETINS      = 14;
static final int MANAGED_ACCTS        = 15;
static final int RECEIVE_FA           = 16;
static final int HISTORICAL_DATA      = 17;
static final int BOND_CONTRACT_DATA   = 18;
static final int SCANNER_PARAMETERS   = 19;
static final int SCANNER_DATA         = 20;
static final int TICK_OPTION_COMPUTATION = 21;

static final int TICK_GENERIC = 45;
static final int TICK_STRING = 46;
static final int TICK_EFP = 47;
static final int CURRENT_TIME = 49;
static final int REAL_TIME_BARS = 50;

```

Note: we manually inserted a blank line after numbers 12 to point up the non-contiguous nature of the listing, differing from that displayed in the sources in question.

6.5.3 Numbering of Tick Types in TickType.java

Interestingly, we can see the tick types supported in TickType.java – As of API version 9.40, the following Request types are enumerated in TickType.java

```
// constants - tick types
public static final int BID_SIZE    = 0;
public static final int BID        = 1;
public static final int ASK        = 2;
public static final int ASK_SIZE   = 3;
public static final int LAST       = 4;
public static final int LAST_SIZE  = 5;
public static final int HIGH       = 6;
public static final int LOW        = 7;
public static final int VOLUME     = 8;
public static final int CLOSE      = 9;
public static final int BID_OPTION = 10;
public static final int ASK_OPTION = 11;
public static final int LAST_OPTION = 12;
public static final int MODEL_OPTION = 13;
public static final int OPEN       = 14;
public static final int LOW_13_WEEK = 15;
public static final int HIGH_13_WEEK = 16;
public static final int LOW_26_WEEK = 17;
public static final int HIGH_26_WEEK = 18;
public static final int LOW_52_WEEK = 19;
public static final int HIGH_52_WEEK = 20;
public static final int AVG_VOLUME  = 21;
public static final int OPEN_INTEREST = 22;
public static final int OPTION_HISTORICAL_VOL = 23;
public static final int OPTION_IMPLIED_VOL = 24;
public static final int OPTION_BID_EXCH = 25;
public static final int OPTION_ASK_EXCH = 26;
public static final int OPTION_CALL_OPEN_INTEREST = 27;
public static final int OPTION_PUT_OPEN_INTEREST = 28;
public static final int OPTION_CALL_VOLUME = 29;
public static final int OPTION_PUT_VOLUME = 30;
public static final int INDEX_FUTURE_PREMIUM = 31;
public static final int BID_EXCH = 32;
```

```

public static final int ASK_EXCH = 33;
public static final int AUCTION_VOLUME = 34;
public static final int AUCTION_PRICE = 35;
public static final int AUCTION_IMBALANCE = 36;
public static final int MARK_PRICE = 37;
public static final int BID_EFP_COMPUTATION = 38;
public static final int ASK_EFP_COMPUTATION = 39;
public static final int LAST_EFP_COMPUTATION = 40;
public static final int OPEN_EFP_COMPUTATION = 41;
public static final int HIGH_EFP_COMPUTATION = 42;
public static final int LOW_EFP_COMPUTATION = 43;
public static final int CLOSE_EFP_COMPUTATION = 44;
public static final int LAST_TIMESTAMP = 45;
public static final int SHORTABLE = 46;

```

6.6 Numbering in rule.c of the shim

We can examine the mapping performed by the shim in the decoding and encoding by looking at `rule.c` in some cases. These examples are pulled from the sources on 10 Jan 2008, and will vary over time.

6.6.1 Numbering of Commands in rule.c

Commands are examined with:

```
grep TagName rule.c | grep 'STV(1,'
```

The output may be sorted, and made more readable by some text transforms:

```

[herrold@centos-5 src]$ grep TagName rule.c | grep 'STV(1,' | \
    sed -e 's/^(w, "/" -e 's/c,.*$//' -e 's/").*(1,/' \
    -e 's/).//' | awk '{print $2"\t"$1}' | sort -n
1      help
2      ping
3      next
4      list
5      wait
6      wake
7      quit
11     verb

```

```

12     news
13     open
14     account
14     acct
15     exec
16     info
17     wild
18     tick
19     book
19     history
20     past
21     report
21     scan
22     ohlc
23     read
24     load
25     bind
26     atonce
27     create
28     modify
29     submit
30     cancel
31     option
[herrold@centos-5 src]$

```

This way we can more easily see the ‘aliased’ commands: `account` and `acct`, and `book` and `history`

6.6.2 Numbering of Requests in rule.c

Requests are examined with:

```
grep TagName rule.c | grep 'STV(2,'
```

And again, the output may be sorted, and made more readable by some text transforms:

```

[herrold@centos-5 src]$ grep TagName rule.c | grep 'STV(2,' | \
    sed -e 's/^.*(w, \"/' -e 's/, x.*$/' \
    -e 's/".*STV(2,/' | awk {'print $2"\t"$1'} | sort -n
1     ReqMktData
2     EndMktData
3     PlaceOrder

```

```
4      CancelOrder
5      OpenOrders
6      AccountData
7      Executions
8      RequestIds
9      ReqConInfo
9      ReqSymInfo
10     ReqMktBook
11     EndMktBook
12     ReqBulletin
13     EndBulletin
14     SetLogLevel
15     AutoOpens
16     AllOpens
17     ManagedAccts
18     FinAdvisor
19     ReplaceFa
20     HistoryReq
21     ExerciseOpts
22     ReqScanSub
23     EndScanSub
24     ReqScanParms
25     EndHistory
50     ReqOhlcSub
51     EndOhlcSub
[herrold@centos-5 src]$
```

6.6.3 Numbering of Messages in rule.c

Messages are examined with:

Not in rule.c

6.6.4 Numbering of Comments in rule.c

Comments are examined with:

Not in rule.c

Part III
Guided Tutorial

We develop a tutorial, working through each implemented command, to provide a usage reference.

Chapter 7

A Tour of Tables

7.1 Tables

7.1.1 Why so many tables

This section will contain more descriptive material about tables

TBD: the initial bulk table load process

TBD: the (few) tables which may be cleaned of obsolete detail by an end user process, and

TBD: the remaining tables of interest normally only to the shim.

TBD: Also adding a backup and restore strategy section is in order

TBD: then a version conversion section is needed.

TBD: RO replication setup of an RO copy for RO operations for scaling to provide local data plant services

7.1.2 What tables are there anyway?

Using the MySQL command line client (“mysql”), and some basic *nix tools, we can produce and inspect a current listing which enumerates all the tables in the shim’s database at any time:

```
[herrold@centos-4 ~]$ echo "show tables ;" | \  
mysql -u rph_shim -h xps400 rph_testing | pr --columns=3
```

2007-09-25 14:12

Page 1

Tables_in_rph_testing	FutDetail	ProductMap
AccountCode	HistoryBar	Protocol
AtomTag	HistoryTag	Rule80A
BarSize	Institution	ScanFilter
Bool	LineItem	SecType
BoxOptions	LocalSet	Stock
Chicago1	Miscellany	SubRequest
ComboLeg	OcaGroup	SubType
ComboSet	OcaType	Symbol
Contract	OptDetail	TickConfig
Country	OrdType	TifType
Currency	OrderAct	Trigger
DepthLimit	OrderFlags	UndType
Duplicates	OrderJournal	Underlying
Exchange	PartialFill	Version
ExecFilter	PastFilter	Volatility

Execution
FinAdvisor

Position

WatchSets

We recall that part of the characterization of the shim is as follows:

The downstream drives the shim by making changes to the database. The shim consults the database at initial startup, and from time to time thereafter when signaled by brief, simple commands. This combination of database and shim integrate persistent database aware storage with the tws. The command interface also serves as an alternative to supplement the existing tws gui interfaces, to permit you to use downstream programs, whether gui or not, to drive the tws through its api.
[\[trading-shim home page\]](#)

7.1.3 Which tables are safe to alter

The shim as part of its initialization process, loads and verifies the consistency of almost all tables, and certain relations between their content. These consistency checks include:

- That the shim version number, contained in table: **Version** is correct for the version of the shim which is running
- That all foreign key dependencies are met
- That all tables from which it reads with `uid` fields are started at: 1, and are have sequential members without gaps; this value is used for some index structures in the shim.

The quick answer is: almost none of the tables may be casually modified, nor re-loaded though manual *ad hoc* efforts, because such tinkering may destroy a required consistency or relationship. This is part of the reason that the initial load scripts are provided.

The tables which may be added to include:

- LOCALSET
- WATCHSETS
- SUBREQUEST

Considering each table in turn, LOCALSET is used in initial and later population of the general shim database with Contract ID's of particular local interest, but which are not included in the refernece tarball as distributed by the trading-shim developers.

WATCHSETS is used to pre-build collections of related symbols, to permit easy projection, for example, into the SUBREQUEST table. Once such a projection done, a large number of tick (see: 3.24) subscriptions may be initiated with a single load (see: 3.14) command. We present an example of this at the dicussion of the load command.

7.1.4 Adding additional tables

It is also possible, and is not prohibited, to add new tables to the database to the set beyond those included with the reference implementation which are included with a tarball from the trading-shim site, of course.

The WATCHSETS table is present to permit an end user have a short-hand way to note and 'remember' security sets of related interest. The shim developers initially populated it from researching common index component members.

Because this information changes over time, sometimes suddenly (thinks of the company underneath 'T' over time - AT&T, Lucent, SBC in recent years; and the sudden disappearance of 'ENR' - Enron), the members for a given WATCHSETS.TAG, that table falls out of date, and is a maintenance burden to some degree.

```
mysql> describe WatchSets ;
```

Field	Type	Null	Key	Default	Extra
uid	int(10) unsigned	NO	PRI	NULL	auto_increment
tag	char(1)	NO			
sub_type	char(4)	NO	MUL		
sec_type	char(4)	NO	MUL		
exch	char(9)	NO	MUL		
name	char(12)	NO			
config	int(10) unsigned	NO			
detail	int(10) unsigned	NO			

```
8 rows in set (0.00 sec)
```

```
mysql> select tag, count(tag) from WatchSets group by tag
        order by tag ;
```

```
+-----+-----+
```

```

| tag | count(tag) |
+-----+-----+
| a   |           1 |
| b   |           1 |
| d   |          19 |
| f   |           2 |
| g   |           1 |
| h   |           2 |
| i   |          94 |
| j   |           3 |
| m   |           1 |
| n   |           3 |
| p   |          15 |
| q   |         162 |
| t   |           2 |
| Y   |           5 |
+-----+-----+
14 rows in set (0.00 sec)

```

The reason one might wish to do so to add a new table, would be to support a new facility for external (what we call: ‘downstream’) code to frame or present results queries in a more human understandable fashion. As our example, we can add a table to decode the WATCHSETS.TAG index values into human meaningful names.

7.1.5 The initial database load process

First we will go through the sequence of relations from lower level tables toward the CONTRACT table in a rapid overview; then we will retrace through descriptions of selected tables, and elaborate on fields.

The scripts, database structure descriptions, and default table load values which ship with every shim tarball are in the `./sql/` subdirectory.

Many of the shim commands require a Contract ID value, which is a shorthand for the CONTRACT.UID for a specific row. That row can be traced back to characteristics of a given security, as its security type (to SECTYPE), its ticker symbol (to SYMBOL), the exchange which is its ‘home’, but which may or may not be the preferred venue at which to trade it which we call its ‘route’ (two uses for the EXCHANGE). Futures add expiration dates (through FUTDETAIL), and Options put and call, and strike prices (through OPTDETAIL).

We initially build this information up progressively in the scripted database load process.

A manual maintenance process

Going forward, one can add to various tables in an *ad hoc* fashion, or by populating the generation tables. The clear downside to ‘one off’ additions is that one has to form all the MySQL `INSERT` statements accurately, and in a fashion which respects ‘foreign key’ constraints, but with the tactical upside that one does not need a deep understanding of the full database to make alterations in a local fork, and most often, in a ‘testing’ database which gets recreated from time to time.

A more automated maintenance process

Once experimentation is over, one can add new ‘tuples’ (as we think of them) to the generation driver tables, and run the creation scripts, and be done.

... Well not completely done. Some data related to transactions like orders and executions, and retrieved history would be lost absent an additional effort to use MySQL’s tools to `mysqldump` and then to restore some tables. If the underlying `schema` of the database has changed, or if numbering of, say, `CONTRACT.UID` has changed [which is unfortunately a common case when symbols have been added to through the first mentioned *ad hoc* manner],

7.1.6 Each starts with the initial database load process

FIXME: more text

1. CURRENCY
2. MISCELLANY
3. STOCK

are used by the load scripts to generate:

UNDERLYING

This is combined with the: `PRODUCTMAP` by the load scripts to generate:

SYMBOL

As there are many rows which are not immediately interesting, the load scripts again use another map: `LOCALSET` to generate:

CONTRACT

The Contract ID is the primary value used to specify a Contract in concise and exact form to most commands.

Bill has noted in an email to the mailing list:

In brief, the preferred way to add a new contract to the `CONTRACT` table is to add an entry to the `LOCALSET[.]` load file

mod/LocalSet.sql, add supporting entries as needed to primary and intermediate load files used to populate SYMBOL, and then recreate the database.

In the worst case, for a new UNDERLYING not yet appearing in the database, you will have to add to one of CURRENCY, MISCELLANY, or STOCK, then, if deriving from that, to PRODUCTMAP, and in any case, add to LOCALSET.

Chapter 8

Working with the database

8.1 The shim database and CONTRACT IDs

The shim is a command-line and dbms controlled interface; as such, we clearly need to consider how to work properly and with facility, with its database, as well as with its commands.

We will go through some sample exercises. We pursue at least a couple of objectives in the context of the shim and using the database of the shim. The exercises will build on one after the other.

We start with some simple database command line operations. These are asking questions about the database with the MySQL `SELECT` query, and performing insertion transactions with the MySQL `INSERT` command.

Then we will re-visit `SELECT` and show a more concise form with the MySQL `LEFT JOIN` clause. Through these exercises, note that we use the more formal `TABLE_NAME.FIELD` qualified form of specifying a match argument in the `WHERE` and `ORDER BY` clauses.

To make the exercise ‘real’, we look at some early tasks we are interested in understanding include:

- how to interpret values (and particularly errors) seen in the logs,
- how to map from a `CONTRACT ID` back into the underlying `SYMBOL`, and
- how to extend the `CONTRACT` (and `FUTDETAIL`) tables.

In coming to this example, we assume the availability of the sample scripts in the shim 070802 release, which is present in the FTP attic. This was a current version at the time this was written, and we have created our examples with the sample `make test` test scripts in the `./bin/` directory.

Almost all of the `uid` values as used by with the shim are arbitrary numbers, representing their sequence of insertion into their owning table. The particular `CONTRACT.UID` we will focus on is with value: 178 used by the `book` and `info` commands in the sample script versions referenced above.

8.1.1 Looking up an underlying SYMBOL from the cid - step by step

The particular use case we are interested in arises from this message series in the shim’s output logging:

```
Aug  2 16:36:01 centos-4 : 6961|59761| 8864983|2|18| 0
                        |book add 15 3;|
Aug  2 16:36:01 centos-4 : 6961|59761| 8865466|2|18| 0
```

```

|book add 178 7;|
Aug  2 16:36:01 centos-4 : 6961|59761| 8865496|3|10| 3
|6|15|3|
Aug  2 16:36:02 centos-4 : 6961|59761| 8885571|3|10| 3
|7|178|7|
Aug  2 16:36:02 centos-4 : 6961|59762| 9001771|3| 4| 2
|      178| 200|No security definition has been found for
the request|
Aug  2 16:36:02 centos-4 : 6961|59762| 9276599|3|12| 1
|      15| 0|0|1| 63.81| 72|bid|insert|STK.SMART.AIG.

```

That message about “No security definition ...” should not be occurring. It indicates that the TWS has concluded, after consulting its upstream state, that there is a problem with a contract ID proposed by the shim.

We can see that some sort of testing in the scripts references a value: 178 and as indicated, this is causing the TWS to return a ‘No security definition’ message. Using `grep` against the sample scripts, we see the occurrence of that 178

```

[herrold@centos-4 shim]$ cd shim_070802
[herrold@centos-4 shim_070802]$ cd bin
[herrold@centos-4 bin]$ grep 178 *
includes: hit_shim 'get contract info'      'info 178 all;\'
includes: hit_shim 'YM market depth'       'book add 178 7;\'
includes: hit_shim 'no market depth'       'book del 178 1;\'
includes.orig: hit_shim 'get contract info' 'info 178 all;\'
includes.orig: hit_shim 'YM market depth'  'book add 178 7;\'
includes.orig: hit_shim 'no market depth'  'book del 178 1;\'
shell:info 178 all;
unsafe: # hit_shim 'get contract info'      'info 178 all;\'
[herrold@centos-4 bin]$

```

How can we fix this? We need to track down what underlying Symbol (actually the particular tradable security it represents) is pointed to by `CONTRACT.UID` value: 178, and determine what the correct value to use is.

It is helpful to determine what the underlying security is. We do can do this step by step with the `mysql` Unix command line MySQL client, as follows:

```

[herrold@centos-4 ~]$ mysql -u rph_shim -h xps400 rph_testing

```

which gets a command prompt, under MySQL account: `rph_shim` on Unix host: `xps400` using database: `rph_testing`. These values are different from

the values in the release script themselves, and indeed we use slightly different keying internally. (Bill uses one set of keying, and Russ, another, so that each developer can use a common MySQL server, simultaneously servicing different databases, to avoid inadvertently changing the other's database contents.)

```
mysql> select * from Contract where Contract.uid = '178';
+-----+-----+-----+-----+-----+
| uid | sid | route | unit | tag |
+-----+-----+-----+-----+-----+
| 178 | 5490 | 18 | 1 | 6 |
+-----+-----+-----+-----+-----+
```

And then we just work across, decoding back up the database hierarchy tree of uid pointers through the corresponding applicable tables.

```
mysql> select * from Symbol where Symbol.uid = '5490';
+-----+-----+-----+-----+-----+-----+-----+
| uid | tid | exch | name | desc | conid |
+-----+-----+-----+-----+-----+-----+
| 5490 | 3 | 6 | YM | DJ IND AVG | MINI | NULL |
+-----+-----+-----+-----+-----+-----+-----+
```

We determine the Exchange it trades on from EXCHANGE.UID = '18'; This is the CONTRACT.ROUTE value. IB uses the virtual exchange: SMART for trades which it may cross (i.e., 'route to') itself. We truncate the EXCHANGE.PRODUCTS entry as the details are not germane here.

```
mysql> select * from Exchange where Exchange.uid = '18';
+-----+-----+-----+-----+-----+-----+
| uid | name | code | desc | products |
+-----+-----+-----+-----+-----+-----+
| 18 | SMART | US | IB SmartRouting | STK,OPT,FUT,IND, ... |
+-----+-----+-----+-----+-----+-----+
```

We can examine the 'home' Exchange at which a CONTRACT is listed with EXCHANGE.UID = '6'. This is the more common expectation as to an Exchange, but IB, as noted above, interjects a potential 'route' for a transaction to occur through local transaction crossing at the SMART exchange.

```
mysql> select * from Exchange where Exchange.uid = '6';
+-----+-----+-----+-----+-----+-----+
| uid | name | code | desc | products |
+-----+-----+-----+-----+-----+-----+
| 6 | ECBOT | US | Electronic CBOT | FUT,FOP |
+-----+-----+-----+-----+-----+-----+
```

The CONTRACT.UNIT field was initially uninteresting to us (it refers to the currency unit of a Contract) as we have USD denominated accounts, studied only USD denominated Contracts, and purchased only the market datastreams supporting those Contract. Accordingly, on the principal of developing while doing continuous testing, we had populated our entries with Contacts we could test, which Contrats trade in that currency. Later as we picked up an interested continental tester, seeking symbols traded on EuroStoxx, we added in a few EUR (CONTRACT.UNIT) denominated Contracts. Still being ‘test driven’ in doing development, we enabled the relevant market and trading rights on one of our IB accounts, and drilled in the DAX and friends, with a little testing.

```
mysql> select * from Currency where Currency.uid = '1';
+-----+-----+-----+-----+-----+
| uid | code | country | floor | plural      |
+-----+-----+-----+-----+-----+
|  1 | USD  | US      |  9    | US Dollars  |
+-----+-----+-----+-----+-----+
```

And now for that mysterious CONTACT.TAG field:

We consciously defered examining the Security Type, which is the only ‘detail’ we needed to extract via the SYMBOL table until this point in our examination. We determine the Type of Security it is, from SYMBOL.TID = ‘3’ which tells us:

```
mysql> select * from SecType where SecType.uid = '3';
+-----+-----+-----+-----+
| uid | type | text | desc  |
+-----+-----+-----+-----+
|  3 | FUT  | FUT  | future |
+-----+-----+-----+-----+
```

A brief aside as to command line database operations:

We happen to know we are dealing with a FUTure here, but it is useful to be able to decode at the command line, the SECTYPE. or indeed the ‘tuple’ to which a given CONTRACT ID refers.

```
$ echo "select SecType.type from Contract          \
      left join Symbol   on Contract.sid   = Symbol.uid   \
      left join SecType  on Symbol.tid     = SecType.uid   \
      where Contract.uid = '178' "              | \
mysql -s -s -r -u rph_shim -h xps400 rph_testing
```

```
FUT
$
```

We can determine the number of rows in the CONTRACT as well:

```
$ echo "select count(uid) from Contract" | \
mysql -s -s -r -u rph_shim -h xps400 rph_testing
205
$
```

For the latter case, we wrote a small script, FIXME - to the appendix - refdecodeCID.sh decodeCID.sh, to simplify typing, and to permit chaining processes together.

```
$ for i in `eq 1 205` ; do ./decodeCID.sh $i ; done
1 CASH.IDEALPRO.USD
2 CASH.IDEALPRO.AUD
3 CASH.IDEALPRO.CAD
...
203 FUT.SMART.GBL.200712
204 FUT.SMART.GBM.200712
205 FUT.SMART.GBS.200712
$
```

which script enumerates the entire set of 'tuples' in the CONTRACT table.

Completing the look-up

And because we know SECTYPE.TYPE = 'FUT', which is a future, we know to use FUTDETAIL for our lookup of the last field in the CONTRACT table, that of the CONTRACT.TAG. We now have the context (telling us to refer to the values in the FUTDETAIL table) to give that tag meaning for the CONTRACT.TAG = '6':

```
mysql> select * from FutDetail where FutDetail.uid = '6';
+-----+-----+-----+
| uid | expiry | multiplier |
+-----+-----+-----+
| 6 | 200706 | 1 |
+-----+-----+-----+
```

The question becomes: Is there a current CONTRACT already present, the CONTRACT.UID value of which we can use in the testing scripts?

That is, is there enough information already in the database to permit us to roll the contract to a new front month? The next front month would end in September 2007. Let's look:

```
mysql> select * from FutDetail where FutDetail.expiry = '200709';
+-----+-----+-----+
| uid | expiry | multiplier |
+-----+-----+-----+
| 7 | 200709 | 1 |
+-----+-----+-----+
```

Putting the Symbol and the expiration tag together, we see that it is there already.

```
mysql> select * from Contract where Contract.sid = '5490'
        and Contract.tag = '7';
+-----+-----+-----+-----+-----+
| uid | sid | route | unit | tag |
+-----+-----+-----+-----+-----+
| 179 | 5490 | 18 | 1 | 7 |
+-----+-----+-----+-----+-----+
```

Looking ahead, we can repeat the process for the next Expiration, in December 2007. Once we know the CONTRACT.TAG, we can also test if that contract is in the database yet:

```
mysql> select * from FutDetail where FutDetail.expiry = '200712';
+-----+-----+-----+
| uid | expiry | multiplier |
+-----+-----+-----+
| 8 | 200712 | 1 |
+-----+-----+-----+
```

```
mysql> select * from Contract where Contract.sid = '5490'
        and Contract.tag = '8';
+-----+-----+-----+-----+-----+
| uid | sid | route | unit | tag |
+-----+-----+-----+-----+-----+
| 180 | 5490 | 18 | 1 | 8 |
+-----+-----+-----+-----+-----+
```

Please note that although the entries in the FUTDETAIL table presently are in sequential order for YM expiration months, this is just an artifact of how that table was initially populated. With MySQL and most databases, the row order position of a given detail line is not material, nor guaranteed. Do not design code which relies on any seemingly 'natural' sequential ordering relation persisting over time, as tables almost certainly become disordered as time passes, and maintenance occurs.

8.1.2 Looking up a underlying SYMBOL from the cid - with LEFT JOIN

The MySQL LEFT JOIN clause construct with SELECT permits a much more concise statement of the information we seek from that single CONTRACT.UID. We use white space and alignment to make it easier to see the clarity of expression which LEFT JOIN confers.

```
mysql> select Contract.sid, SecType.type, Exchange.name,
           Symbol.name, FutDetail.expiry from Contract
left join Symbol    on Contract.sid = Symbol.uid
left join SecType   on Symbol.tid   = SecType.uid
left join Exchange  on Contract.route = Exchange.uid
left join FutDetail on Contract.tag  = FutDetail.uid
           where Contract.uid = '178';
+-----+-----+-----+-----+-----+
| sid | type | name | name | expiry |
+-----+-----+-----+-----+-----+
| 5490 | FUT  | SMART | YM   | 200706 |
+-----+-----+-----+-----+-----+
```

Note: As a quick side note as to argument order in MySQL LEFT JOIN matching: There is no sensitivity as to the right and left hand side of a match clause – that is:

```
left join FutDetail on Contract.tag = FutDetail.uid
```

will produce the same result as:

```
left join FutDetail on FutDetail.uid = Contract.tag
```

and so forth. As a matter of building up the queries, it is visually simpler to keep all the .uid match parts to the right, but we may from time to time inadvertently swap the sequence, because argument order is not material.

Let's confirm that look-up with the decoding script:

```
$ ./decodeCID.sh 178
178 FUT.SMART.YM.200706
$
```

This example was from some foreknowledge – Recall from our previous example that we were looking at a Future, and so we added the clause:

```
left join FutDetail on Contract.tag = FutDetail.uid
```

Stock Lookup

The Stock case is simpler:

```
mysql> select Contract.sid, SecType.type, Exchange.name,
           Symbol.name from Contract
           left join Symbol on Contract.sid = Symbol.uid
           left join SecType on Symbol.tid = SecType.uid
           left join Exchange on Contract.route = Exchange.uid
           where Contract.uid = '15';
```

```
+-----+-----+-----+-----+
| sid | type | name | name |
+-----+-----+-----+-----+
| 2850 | STK | SMART | AIG |
+-----+-----+-----+-----+
```

In the case of a Stock, we were able to omit two parts – the FUTDETAIL.EXPIRY and the left join FutDetail on Contract.tag = FutDetail.uid parts, as a stock is perpetual, or a least has no stated expiration.

Index symbol Lookup

The Index (SecType.type: IND) case is similar. Let's dig out a common one, walking the other way, from SYMBOL to CONTRACT, and then back to show the LEFT JOIN's power:

```
mysql> select * from Symbol where Symbol.name like 'TICK%';
+-----+-----+-----+-----+-----+-----+-----+-----+
| uid | tid | exch | name      | desc                | conid |
+-----+-----+-----+-----+-----+-----+-----+
| 12  | 4   | 2    | TICK-NYSE | ADVANCE - DECLINE  | NULL  |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
mysql> select * from Contract where Contract.sid = '12';
+-----+-----+-----+-----+-----+
| uid | sid | route | unit | tag |
+-----+-----+-----+-----+-----+
| 172 | 12  | 18    | 1    | 0   |
+-----+-----+-----+-----+-----+
```

So we suspect CONTRACT.UID = 172 is our candidate. Let's verify that.

```
mysql> select Contract.sid, SecType.type, Exchange.name,
           Symbol.name from Contract
```

```

left join Symbol    on Contract.sid    = Symbol.uid
left join SecType   on Symbol.tid      = SecType.uid
left join Exchange  on Contract.route  = Exchange.uid
      where Contract.uid = '172';

```

```

+-----+-----+-----+-----+
| sid | type | name  | name      |
+-----+-----+-----+-----+
|  12 | IND  | SMART | TICK-NYSE |
+-----+-----+-----+-----+

```

Currency Lookup

And Currencies (called forex, for 'Foreign Exchange'; in the IB TWS GUI, SECTYPE.TYPE: CASH) – now that we know the technique:

```
mysql> select * from Symbol where Symbol.name like 'GBP%';
```

```

+-----+-----+-----+-----+-----+-----+-----+
| uid | tid | exch | name | desc | conid |
+-----+-----+-----+-----+-----+-----+
|   6 |  6 |   9 | GBP | GB Pounds | NULL |
| 3544 |  1 |   2 | GBP | GABLES RESIDENTIA | NULL |
| 5541 |  3 |   5 | GBP | GB Pounds | NULL |
| 5554 |  2 |   5 | GBP | GB Pounds | NULL |
+-----+-----+-----+-----+-----+-----+

```

4 rows in set (0.01 sec)

(Note here: that the % character is a wildcard match indicator in MySQL, and is commonly used in a MySQL LIKE clause.)

```
mysql> select * from Contract where Contract.sid = '6';
```

```

+-----+-----+-----+-----+-----+
| uid | sid | route | unit | tag |
+-----+-----+-----+-----+-----+
|  6 |  6 |   9 |   1 |  0 |
+-----+-----+-----+-----+-----+

```

```
mysql> select Contract.sid, SecType.type, Exchange.name,
      Symbol.name from Contract
left join Symbol    on Contract.sid    = Symbol.uid
left join SecType   on Symbol.tid      = SecType.uid
left join Exchange  on Contract.route  = Exchange.uid
      where Contract.uid = '6';

```

```

+-----+-----+-----+-----+-----+
144

```

sid	type	name	name
6	CASH	IDEALPRO	GBP

And as expected, one can find:

CASH.IDEALPRO.GBP

IDEALPRO is IB's in house foreign exchange crossing facility.

Remainder case Lookup

Note: As of August 2007 we have not developed with tests for some security types. Indeed, we do not even know for a certainty that there is, or is not a way for some to be accessed through the API, as it is perfectly possible (indeed, from the GUI client, it seems likely) that there are non-public interfaces to the upstream for certain parameters.

Particularly, consider the full list of security types:

```
mysql> select * from SecType ;
```

uid	type	text	desc
1	STK	STK	stock
2	OPT	OPT	option
3	FUT	FUT	future
4	IND	IND	index
5	FOP	FOP	option on future
6	CASH	CASH	cash (ideal FX)
7	BOND	BOND	bond
8	BAG	BAG	combination order

For future options (SEC_TYPE.TYPE: FOP), future spreads (a type we omit from the SEC_TYPE table), options (SEC_TYPE.TYPE: OPT), warrants (a type we omit from the SEC_TYPE table), or bonds (SEC_TYPE.TYPE: BOND), as the specification is more complex, and these are not something our research needs presently encompass. Neither have we explored the combination order (SEC_TYPE.TYPE: BAG) to any material extent recently.

If we pick up a committed tester or two for these security types, we are certainly willing to discuss adding this to our development.

8.1.3 Looking up a CONTRACT.UID with LEFT JOIN

With our new knowledge about the power of the LEFT JOIN clause, it becomes straightforward to look up the CONTRACT.UID (in shorthand, a CID) to use for various security types:

1. Currency Lookup - LEFT JOIN

A Currency (here: CASH.IDEALPRO.AUD):

```
mysql> select Contract.uid, Contract.sid, SecType.type,
             Exchange.name, Symbol.name from Contract
             left join Symbol    on Contract.sid    = Symbol.uid
             left join SecType  on Symbol.tid      = SecType.uid
             left join Exchange on Contract.route  = Exchange.uid
       where SecType.type = 'CASH' and
             Exchange.name = 'IDEALPRO' and
             Symbol.name   = 'AUD' ;
```

uid	sid	type	name	name
2	2	CASH	IDEALPRO	AUD

2. Stock Lookup - LEFT JOIN

A Stock (here: STK.SMART.IBM):

```
mysql> select Contract.uid, Contract.sid, SecType.type,
             Exchange.name, Symbol.name from Contract
             left join Symbol    on Contract.sid    = Symbol.uid
             left join SecType  on Symbol.tid      = SecType.uid
             left join Exchange on Contract.route  = Exchange.uid
       where SecType.type = 'STK' and
             Exchange.name = 'SMART' and
             Symbol.name   = 'IBM' ;
```

uid	sid	type	name	name
84	3718	STK	SMART	IBM

3. Index symbol Lookup - LEFT JOIN

An Index symbol (here: IND.SMART.TRIN-NYSE):

```
mysql> select  Contract.uid, Contract.sid, SecType.type,
              Exchange.name, Symbol.name from Contract
              left join Symbol    on Contract.sid    = Symbol.uid
              left join SecType   on Symbol.tid     = SecType.uid
              left join Exchange  on Contract.route = Exchange.uid
              where SecType.type = 'IND' and
                 Exchange.name = 'SMART' and
                 Symbol.name   = 'TRIN-NYSE';
```

uid	sid	type	name	name
171	11	IND	SMART	TRIN-NYSE

4. Future Lookup - LEFT JOIN

... and finally, a Future (here: FUT.SMART.YM.200703):

```
mysql> select  Contract.uid, Contract.sid, SecType.type, Exchange.name,
              Symbol.name, FutDetail.expiry from Contract
              left join Symbol    on Contract.sid    = Symbol.uid
              left join SecType   on Symbol.tid     = SecType.uid
              left join Exchange  on Contract.route = Exchange.uid
              left join FutDetail on Contract.tag   = FutDetail.uid
              where SecType.type = 'FUT' and
                 Exchange.name  = 'SMART' and
                 Symbol.name    = 'YM' and
                 FutDetail.expiry = '200703' ;
```

uid	sid	type	name	name	expiry
177	5490	FUT	SMART	YM	200703

8.1.4 Adding a new underlying SYMBOL to the CONTRACT table

As a hypothetical (this example being based on the tarball of 2 August 2007 [‘shim-070802.tgz’]), and knowing that Expiration dates will continue to roll with the passage of time, say a couple of years had passed and we need to handle the new ‘front month’ March 2009 expiration for FUT.ECBOT.YM. This instantiation of the security with a new expiration month is treated as a new CONTRACT to trade at IB, and so calls for addition of a new CONTRACT in the database as well.

```
mysql> select * from FutDetail where FutDetail.expiry = '200903';  
Empty set (0.00 sec)
```

Which is saying that it is not yet a known front month.

Let’s see what we do have then:

```
mysql> select * from FutDetail order by FutDetail.expiry ;  
+-----+-----+-----+  
| uid | expiry | multiplier |  
+-----+-----+-----+  
| 1 | 200603 | 1 |  
| 2 | 200606 | 1 |  
| 3 | 200609 | 1 |  
| 4 | 200612 | 1 |  
| 5 | 200703 | 1 |  
| 6 | 200706 | 1 |  
| 7 | 200709 | 1 |  
| 8 | 200712 | 1 |  
| 9 | 200803 | 1 |  
| 10 | 200806 | 1 |  
| 11 | 200809 | 1 |  
| 12 | 200812 | 1 |  
+-----+-----+-----+
```

NOTE: This next process, of adding information to the database, is most safely done when the shim is stopped. The shim can accomodate ‘live additions’ to most of its database tables, and can gain awareness of them with the `load;` command. It is safer to not get into the habit of doing additions on a live copy, since one might forget to send the `load;`, and become mystified as to why something is not working as one expects.

As a matter of database maintenance, once orders are pending or filled against a given CONTRACT.UID, or any subordinate table pointed to by it, such as , here, FUTDETAIL.UID, any rows present must be retained until all

subordinate 'foreign keys' pointing at it, through the CONTRACT table are purged. This is because each subordinate key entry of 'foreign key' table members in the chain must be present to ensure a complete and consistent database exists for the shim.

This is in turn required due to the reliance of the shim, in its C++ code upon both strong 'foreign key' enforcement, and also on the shim's need for numerically contiguous uid entries in some tables.

MySQL per userid account rights

Additionally, as a matter of data security, we have a chance here to show that some userid accounts run with lesser privileges than others. Recall that we logged in thus:

```
[herrold@centos-4 ~]$ mysql -u rph_shim -h xps400 rph_testing
```

Let's try an INSERT while connected in the rph_shim userid:

```
mysql> insert into FutDetail set expiry = '200903', multiplier = '1';
ERROR 1142 (42000): INSERT command denied to user
'rph_shim'@'centos-4.first.lan' for table 'FutDetail'
```

This denial is good, for it shows that our intentionally low privilege user cannot inadvertently damage some needed integrity of the database. In the reference client, the user code is used for such operations needing broader rights. We log out, and then back in with the new userid bearing the needed permissions.

```
[herrold@centos-4 ~]$ mysql -u rph_code -h xps400 rph_testing
```

and repeat the attempted INSERT.

```
mysql> insert into FutDetail set expiry = '200903', multiplier = '1';
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from FutDetail where FutDetail.expiry = '200903';
+-----+-----+-----+
| uid | expiry | multiplier |
+-----+-----+-----+
| 13 | 200903 |          1 |
+-----+-----+-----+
```

Recall that from the table definition, that FUTDETAIL.UID entries are assigned sequentially and contiguously by the database engine. As a general rule, all uid values are sequentially and contiguously assigned by the database server backend, and all so assigned are in turn expected by the shim to be in that form.

The database engine would have prevented a 'malformed':

[“no 'foreign key' yet existed ”]

line with the value: 13 in the CONTRACT.TAG field. Let's confirm this:

```
mysql> select * from Contract where Contract.tag = '13';
Empty set (0.00 sec)
```

Extending the CONTRACT table - part 1

Recall where we are: We have all information we need from a prior FUT.SMART.YM to know the Symbol (Contract.sid: 5490), Exchange route (Contract.route: 18), and unit (Contract.unit: 1) and will hard code it. This is a less portable than some other approaches, but will suffice for our first example involving adding to the CONTRACT table.

Recall that we are in the code reference user account, as we are doing an INSERT:

```
mysql> insert into Contract set sid = '5490', route = '18',
      unit = '1', tag = '13';
Query OK, 1 row affected (0.01 sec)
```

And then back switching into the unprivileged user, under the familiar data security principle of only using the 'least privilege' required for an operation:

```
mysql> select * from Contract where Contract.tag = '13';
```

uid	sid	route	unit	tag
185	5490	18	1	13

```
mysql> select * from Contract;
```

uid	sid	route	unit	tag
1	1	9	1	0

...

```

| 183 | 5490 |    18 |    1 | 11 |
| 184 | 5490 |    18 |    1 | 12 |
| 185 | 5490 |    18 |    1 | 13 |
+-----+-----+-----+-----+-----+

```

And now we can refer to a `CONTRACT.UID` of 185 in transactions as the shorthand for:

```
FUT.ECBOT.YM expiration 200903
```

A similar, but a simpler, analysis applies to Stocks and so forth, and the process to add new Symbols, and thence Contracts. Obviously the database has ‘foreign key’ constraints which must be honored, and so the database administrator needs to observe some care as to unloading and renumbering, or more complexly, merging in changes between the upstream reference database and the local working copy. This is out of scope here, but we highlight the matter for local case analysis.

8.1.5 Fixing the make test

And of course the impetus for this discussion was to fix the broken `make test`, which as it turned out, was referring to a stale contract. We needed to change the 178 to 179 in a couple scripts.

We will use the Unix `sed` command for this purpose, as it is designed to do quick in-place edits without the overhead of opening a full blown edit client.

Determine the files to edit with `grep`:

```

[herrold@centos-4 shim]$ cd shim_070802
[herrold@centos-4 shim_070802]$ cd bin
[herrold@centos-4 bin]$ grep 178 *
includes:    hit_shim 'get contract info'      'info 178 all;'
includes:    hit_shim 'YM market depth'       'book add 178 7;'
includes:    hit_shim 'no market depth'       'book del 178 1;'
includes.orig: hit_shim 'get contract info'    'info 178 all;'
includes.orig: hit_shim 'YM market depth'     'book add 178 7;'
includes.orig: hit_shim 'no market depth'     'book del 178 1;'
shell:info 178 all;
unsafe:  # hit_shim 'get contract info'      'info 178 all;'

```

Do the edits with a modern implementation of `sed` [we expect the ‘-i’ option to be present here, and it seems that Apple’s OS/X 10.4 Xcode development environment does not have such]:

```
[herrold@centos-4 bin]$ sed -i -e 's/178/179/' includes
[herrold@centos-4 bin]$ sed -i -e 's/178/179/' shell
[herrold@centos-4 bin]$ sed -i -e 's/178/179/' unsafe
```

Verify that the edits are done (note that some files already had 179 in them so more lines show up the second time we look with grep).

```
[herrold@centos-4 bin]$ grep 179 *
includes: hit_shim 'get contract info' 'info 179 all;'
includes: hit_shim 'YM market depth' 'book add 179 7;'
includes: hit_shim 'no market depth' 'book del 179 1;'
includes: hit_shim 'YM history query' 'past add 179 11;'
        # 4:5; 5:6; 6:8; 7:11; 8:13; 9:16
includes.orig: hit_shim 'YM history query' 'past add 179 11;'
        # 4:5; 5:6; 6:8; 7:11; 8:13; 9:16
periodic: hit_shim '' 'past add 179 11;'
        ; sleep30
shell:info 179 all;
unsafe: # hit_shim 'get contract info' 'info 179 all;'
[herrold@centos-4 bin]$
[herrold@centos-4 bin]$ cd ..
```

And rerun the test:

```
[herrold@centos-4 shim_070802]$ make test
```

We show the whole test log sequence here:

```
Aug  3 14:08:08 centos-4 : 21927|50887| 2592233|4|100| 5
        |# |4|100|5|*****|
Aug  3 14:08:08 centos-4 : 21927|50887| 2592240|4|100| 5
        |# |4|100|5|0.31|999999|data|
Aug  3 14:08:08 centos-4 : 21927|50887| 2592244|4|100| 5
        |# |4|100|5|*****|
Aug  3 14:08:08 centos-4 : 21927|50887| 2592284|3| 9| 1|1|
Aug  3 14:08:08 centos-4 : 21927|50888| 2836686|3| 4| 2
        | -1|2104|Market data farm connection is OK:usfuture|
Aug  3 14:08:08 centos-4 : 21927|50888| 2836705|3| 4| 2
        | -1|2104|Market data farm connection is OK:usfarm|
Aug  3 14:08:08 centos-4 : 21927|50888| 2836740|2|11| 0|verb Detail;|
Aug  3 14:08:08 centos-4 : 21927|50888| 2856466|3|14| 1|5|
Aug  3 14:08:09 centos-4 : 21927|50889| 3826240|2|19| 0
        |past add 179 11;|
```

```

Aug 3 14:08:09 centos-4 : 21927|50889| 3846936|3|20| 3|1|179|11|
Aug 3 14:08:09 centos-4 : 21927|50889| 4582936|3| 4| 2
|      -1|2106|HMDS data farm connection is OK:ushmds2a|
Aug 3 14:08:09 centos-4 : 21927|50889| 4588531|3| 4| 2
|      179| 165|Historical Market Data Service query
message:HMDS server connection was successful.|
Aug 3 14:08:11 centos-4 : 21927|50890| 4830434|3|17| 3|      179|7|
Aug 3 14:08:11 centos-4 : 21927|50890| 4830300|3| 1| 1
|20070803 14:07:40|13487.0|13487.0|13487.0|13487.0|      8
|13487.0|false|FUT.SMART.YM.
Aug 3 14:08:11 centos-4 : 21927|50890| 4830322|3| 1| 1
|20070803 14:07:45|13486.0|13490.0|13486.0|13489.0|      44
|13488.0|false|FUT.SMART.YM.
Aug 3 14:08:11 centos-4 : 21927|50890| 4830343|3| 1| 1|20070803 14:07
:50|13489.0|13489.0|13486.0|13486.0|      19|13488.0|false
|FUT.SMART.YM.
Aug 3 14:08:11 centos-4 : 21927|50890| 4830364|3| 1| 1|20070803 14:07
:55|13485.0|13486.0|13485.0|13485.0|      22|13485.0|false
|FUT.SMART.YM.
Aug 3 14:08:11 centos-4 : 21927|50890| 4830384|3| 1| 1|20070803 14:08
:00|13486.0|13486.0|13484.0|13484.0|      9|13485.0|false
|FUT.SMART.YM.
Aug 3 14:08:11 centos-4 : 21927|50890| 4830405|3| 1| 1|20070803 14:08
:05|13484.0|13484.0|13480.0|13481.0|     151|13482.0|false
|FUT.SMART.YM.
Aug 3 14:08:11 centos-4 : 21927|50890| 4830426|3| 1| 1|20070803 14:08
:08|13481.0|13481.0|13478.0|13478.0|      34|13479.0|false
|FUT.SMART.YM.
Aug 3 14:08:11 centos-4 : 21927|50890| 4833784|4|100| 5|# |4
|100|5|event: history insert|(179, 2, 20070803 14:07:
40 -- 20070803 14:08:08)|
Aug 3 14:08:13 centos-4 : 21927|50893| 7818519|2|12| 0|news on all;|
Aug 3 14:08:13 centos-4 : 21927|50893| 7818659|2|12| 0|news off all;|
Aug 3 14:08:13 centos-4 : 21927|50893| 7818676|3|12| 1|all|
Aug 3 14:08:13 centos-4 : 21927|50893| 7818811|2|14| 0|acct on;|
Aug 3 14:08:13 centos-4 : 21927|50893| 7818994|2|15| 0|info 15 new;|
Aug 3 14:08:13 centos-4 : 21927|50893| 7819101|2|15| 0|info 178 all;|
Aug 3 14:08:13 centos-4 : 21927|50893| 7819208|2|17| 0|tick add 15 1;|
Aug 3 14:08:13 centos-4 : 21927|50893| 7840445|3|13| 1|
Aug 3 14:08:13 centos-4 : 21927|50893| 7861111|3| 6| 2|on|
Aug 3 14:08:13 centos-4 : 21927|50893| 7902311|3| 8| 1|14:05|

```

Note that there is no longer an error as to:

—No security definition has been found for the request—

We have met our goal of fixing an obsolete reference, which was causing error noise in the log file, and also have learned about adding new expirations and indeed, underlying securities to obtain market data, history, and to trade.

8.1.6 Tabular database table listings in other contexts

We had a request to demonstrate how to emit a tabular listing of information held in the database from the command line. The user wished to produce a simple listing to post at a website, without having to update the listing manually from time to time.

```
[herrold@centos-4 docs]$ echo "select * from Exchange \
      order by 'desc' " | mysql-u rph_shim -h xps400 rph_testing
uid      name      code      desc      products
3        AMEX      US        American Stock Exchange STK,OPT,IND
28       ARCA      US        Archipelago      STK
30       BTRADE   US        Bloomberg Tradebook      STK
      ...
38       WINNER   GB        Winterflood Securities Ltd      STK
46       IBIS     DE        XETRA: eXch electronic TRAding STK,IND
[herrold@centos-4 docs]$
```

and as the question was asked in a web server context, it is possible to emit a well-formed HTML static table fragment:

```
[herrold@centos-4 docs]$ cat ./showExchangeTable.php
#!/usr/bin/php -qc/etc
<?php
//      Copyright (c) 2007 Owl River Company
//      ALL rights reserved ; unauthorized use prohibited
//      info@owlriver.com
//
$debug = "y";
$debug = "";
//
//      get the SQL passwords
include './include.inc';
//
if (" $debug" != "") {
    print "-|" . $sql_server . "|-" .
    $sql_user . "|-" .
    $sql_passwd . "|-" .
    154
```

```

    $sql_name . "|-|" .
    "|-\n";
    print "<hr>\n";
}
//      Connect to the server
$link = mysql_connect("$sql_server", "$sql_user", "$sql_passwd")
    or die("Could not connect");
if ("$debug" != "") {
    print "Connected successfully to host: $sql_server<br>\n";
}
//      and Verify the database is accessible
$isdb = mysql_select_db("$sql_name")
    or die ("Error connecting to database");
if ("$debug" != "") {
    print "Connected successfully to database: $sql_name<br>\n";
}
//
////////////////////////////////////
//
print "<hr>\n";
//
//      This is the table we wish to dump the contents of
$select1 = "select * from Exchange order by 'desc'";
$result1 = mysql_query($select1)
    or $mysql_eval_error = mysql_error();
if ($mysql_eval_error) {
    print "mysql_error 1: $mysql_eval_error \n";
}
//
//      Did we find any rows
$numrow1 = mysql_num_rows($result1);
if ("$debug" != "") {
    print "numrow1: $numrow1 <br>";
}
if ($numrow1 > 0 ) {
//
//      If so, print the table
print "<table>";
print "<tr><td>uid</td><td>name</td><td>code</td>
<td>desc</td><td>products</td></tr>\n";
while ($row1 = mysql_fetch_assoc($result1)) {
//      list items are: uid      name      code      desc      products

```

```

        $li_uid      = $row1["uid"];
        $li_name     = $row1["name"];
        $li_code     = $row1["code"];
        $li_desc     = $row1["desc"];
        $li_products = $row1["products"];
        print "<tr><td>$li_uid</td><td>$li_name</td><td>$li_code</td>
<td>$li_desc</td><td>$li_products</td></tr>\n";
    }
//
    print "</table>\n";
    print $numrow1 . " rows found<br>\n";
}
//
print "<hr>\n";
//
?>

```

The file shown above is in PHP command line interface ('CLI') format with the initial `#!/usr/bin/php -qc/etc` line; if one were building a web interface, by removing that initial line, this code would also work as for small tables. Note that it tests the number of returned rows, such that a multi-page web display interface might be accomodated with just a bit more code.

The CLI version will yield a result when run at the command line, which is simple to capture into a file using common shell `stdout` redirect operators:

```

[herrold@centos-4 docs]$ ./showExchangeTable.php
<hr>
<table><tr><td>uid</td><td>name</td><td>code</td>
  <td>desc</td><td>products</td></tr>
<tr><td>3</td><td>AMEX</td><td>US</td>
  <td>American Stock Exchange</td><td>STK,OPT,IND</td></tr>
<tr><td>28</td><td>ARCA</td><td>US</td>
  <td>Archipelago</td><td>STK</td></tr>
<tr><td>30</td><td>BTRADE</td><td>US</td>
  <td>Bloomberg Tradebook</td><td>STK</td></tr>
  ...
<tr><td>38</td><td>WINNER</td><td>GB</td>
  <td>Winterflood Securities Ltd</td><td>STK</td></tr>
<tr><td>46</td><td>IBIS</td><td>DE</td>
  <td>XETRA: eXch electronic TRAding</td><td>STK,IND</td></tr>
</table>
54 rows found<br>
<hr>

```

```
[herrold@centos-4 docs]$
```

The database keying file is trivial in form:

```
[herrold@centos-4 docs]$ cat ./include.inc
<?php
$sql_server = "xps400";
$sql_user = "rph_shim";
$sql_passwd = "";
$sql_name = "rph_testing";
?>
[herrold@centos-4 docs]$
```

FIXME: subsection

8.1.7 How to extend the Symbol (and then CONTRACT) tables

The IRC channel (`#interactivebrokers`, on the `irc.othernet.org` servers) had this question (we truncate the IRC user nicknames):

```
10:35 +Mixx> I need ib ticker quote help ok to ask here?
10:35 +Mixx> AD-NYSE
10:35 +Mixx> VOL-NYSE
10:35 +Mixx> i get no quotes
10:40 +Boxx> same here
10:42 +Mixx> ok
10:42 +Mixx> IRT software is getting quotes on it
           through tws i see so ok here
10:42 +Mixx> thx
10:43 +Boxx> are you sure it's using the same symbols you typed ?
11:34 +Mixx> y
```

and so we check to see if we presently support framing tick or past commands on those Indices.

```
mysql> select Contract.uid, Contract.sid, SecType.type,
           Exch.name as route, Exchange.name as home,
           Symbol.name from Contract
           left join Symbol          on Contract.sid = Symbol.uid
           left join SecType         on Symbol.tid  = SecType.uid
           left join Exchange as Exch on Contract.route = Exch.uid
           left join Exchange        on Symbol.exch = Exchange.uid
           where SecType.type = 'IND' and Symbol.name like '%NYSE%';
+-----+-----+-----+-----+-----+-----+
| uid | sid | type | route | home | name      |
+-----+-----+-----+-----+-----+-----+
| 172 |  12 | IND  | SMART | NYSE | TICK-NYSE |
| 171 |  11 | IND  | SMART | NYSE | TRIN-NYSE |
+-----+-----+-----+-----+-----+-----+
```

Note: that we had to ‘alias’ the `EXCHANGE` table to do the projection to permit seeing both the order ‘route’ Exchange of the `Contract` and the ‘home’ Exchange of the `Symbol` in a single query.

The answer is: the ultimate `CONTRACT.UID` values we want to use are not currently present, so we would need to add them to the `CONTRACT` table in the proper form, before we can test if the feeds are live.

Digging further, are they even in the SYMBOL table yet?

```
mysql> select Symbol.uid, Symbol.name from Symbol
       where Symbol.name like '%NYSE%';
```

```
+-----+-----+
| uid | name      |
+-----+-----+
|  12 | TICK-NYSE |
|  11 | TRIN-NYSE |
+-----+-----+
```

and again the answer is in the negative, so we will also need to first add these indices to the SYMBOL table as well.

To add these, all of the non-optional fields in the SYMBOL table need to be specified. The non-optional, or mandatory fields are: `tid`, `exch`, and `name`. The two remaining fields: `desc`, and `conid` are optional, in the sense that no sub-table relation mandates ‘sensible’ contents for them.

As we are undertaking role of a maintainer of local dataset extensions, however, we probably want to track IB’s name assignments for these fields. We started to perform minimal maintainer duties previously with the simplified case of adding a new front month for `FUT.SMART.YM` 8.1.4. We need to look a bit deeper now.

Populating the SYMBOL table manually

One way to test would be to use the TWS GUI, and seek to add each tuple. The TWS communicates upstream to IB, and only offers sub menu picks in turn of potentially ‘correct’ contracts.

Using the TWS GUI, `Contract Info — Description`, we get a pop-up box, and it seems: `AD-NYSE`, an Index, is at `NYSE`, or in our tuple notation from the TWS GUI as follows: `IND.NYSE.AD-NYSE` It is not immediately obvious that the route is `SMART` on this index; in earlier experimentation, we determined that this works experiemntally, when we added `IND.NYSE.TICK-NYSE` and `IND.NYSE.TRIN-NYSE`.

See Figure 8.1

Using the TWS GUI, `Contract Info — Contract Details`, it seeks to open a web page at the following URL (as it is quite long, we have broken it over three lines; obviously a URL needs to be ‘reassembled’ back onto a contiguous line with no intervening whitespace to work in a web browser or a web page ‘screen scraper’ tool):

```
http://www.interactivebrokers.co.uk/contract_info/index.php?
    action=Details&site=IB&conid=33887584&detlev=2
    &sess=1192638048
```

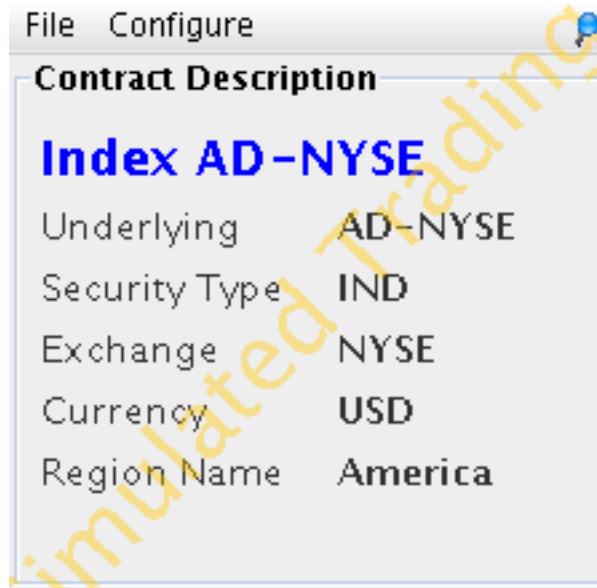


Figure 8.1: Contract Info — Description

In experimenting, we find that the last line with the `sess` value is not strictly needed. The other four HTTP GET variables and values, and our inferred meaning for each are:

1. `action=Details` - a common and conventional HTTP GET action FORM variable name for indicating a FORM SUBMIT response is sought from the webserver. Of course this is a common way to implement a Remote Procedure Call (“RPC”) [here, obtaining extended CONTRACT DETAILS] in a now customary form, which may be proxied, or directly be permitted to traverse many firewalls on tcp/80.
2. `site=IB` - a website ‘badging’ capability, for so called ‘white box’ re-branding by IB marketing partners.
3. `conid=33887584` - the IB Conid; a Contract identifier, specific to IB, which our observation indicates does not vary often, if at all (we have not seen one change yet, but are aware of no IB representation on this), as to a specific tradeable security.
4. `detlev=2` - Detail level. This is self-explanatory, but we are aware of no IB documentation of allowable values.

Note that on the webpage returned that the following ‘boilerplate’ warning is also present at the bottom of the page, outside the screenshot;

File Edit View Go Bookmarks Tools Window Help

Home
 Bookmarks
 Google
 Trading-shim Homepage
 LibraryTh

xps40...
 Gmail...
 Googl...
 SQL ...
 Proce...
 My lib...

General Information

Name	NYSE ADVANCE DECLINE INDEX
TWS Symbol	AD-NYSE
Contract Type	IND
Currency	USD
Country of Origin	Unknown (XX)
RLA	0
RLM	0

Available Exchanges

Exchanges	NYSE
-----------	----------------------

Contract Identifiers

Conid	33887584
-------	----------

NEW YORK STOCK EXCHANGE (NYSE) | [Top](#)

Local Name	AD-NYSE
Local Class	-
Price Parameters	161 Range price > 0
Size Parameters	Minimum Order Size 1 Order S

The information and materials provided via the Interactive Brokers Contract Information Center are provided "as is" and without warranties of any kind as to the accuracy or validity. Additionally, IB provides links to other sites that are not maintained by IB. IB does not endorse those sites and is not responsible for the content of such other sites. Not all contracts are available for all account types.

We concur that this is not wholly reliable information, and note in passing that a mailing list correspondent stated:

```
> But I _am_ concerned that a MKT order took 40 minutes to fill
> on the initial SELL. What are the 'Regular trading hours' on
> the exchange upon which 'GBM' trades?
```

```
08:00 - 22:00 (CEST)
```

```
>>> From the IB site, the relevant exchange hours seem to be:
>>> 07:30 - 20:00 (CET)
>>> www.eurexchange.com
```

```
this information is outdated...
```

Obviously it is an almost impossible clerical work load, whether for 'Interactive Brokers Contract Information Center' or a shim developer or user, or the author of this work to keep manually maintained systems wholly current.

Populating the SYMBOL table with automation

Manual processes are of course slow and subject to clerical error.

QUERY: Will the wild command permit us to gather a collection of matching conid values, to then ask upstream to IB directly?

Populating the SYMBOL table

FIXME With some handwaving, we assume we have the relevant conid in hand, and have performed the webscrape, which yields SYMBOL.DESCR = 'NYSE ADVANCE DECLINE INDEX', for SYMBOL.CONID = '33887584'. We have now gathered all we need to do the INSERT of a new row in the Symbol table.

First let's test to see what it will look like:

```
mysql> select SecType.uid as tid, Exchange.uid as exch,
         'AD-NYSE' as name,
```

```

'NYSE ADVANCE DECLINE INDEX' as 'desc',
'33887584' as conid from SecType
left join Exchange on Exchange.name = 'NYSE'
where SecType.type = 'IND' and
Exchange.name = 'NYSE' limit 1 ;

```

```

+-----+-----+-----+-----+-----+
| tid |  exch | name   | desc                                     | conid |
+-----+-----+-----+-----+-----+
|   4 |     2 | AD-NYSE | NYSE ADVANCE DECLINE INDEX | 33887584 |
+-----+-----+-----+-----+-----+

```

which are the values we hoped to see; As previously discussed at [8.1.4](#) 'MySQL per userid account rights', we need to switch into the code MySQL userid to get INSERT rights.

Let's examine some status variables concerning the MySQL client connection which we are using:

```

mysql> \s
-----
mysql  Ver 14.7 Distrib 4.1.20, for redhat-linux-gnu
      (i686) using readline 4.3

Connection id:          893
Current database:      rph_testing
Current user:          rph_code@centos-4.first.lan
...
Connection:           xps400 via TCP/IP
...
Uptime:               12 days 5 hours 1 min 56 sec

Threads: 1  Questions: 36699  Slow queries: 0  Opens: 49
          Flush tables: 1  Opentables: 64  Queries per second avg: 0.035
-----

```

Relevant to our concerns, we note that the connection is in the: rph_code user account rights.

```

mysql> insert into Symbol (tid, exch, name, Symbol.desc, conid)
select SecType.uid as tid, Exchange.uid as exch,
      'AD-NYSE' as name,
      'NYSE ADVANCE DECLINE INDEX' as 'desc',
      '33887584' as conid from SecType
left join Exchange on Exchange.name = 'NYSE'

```

```

        where SecType.type = 'IND' and
        Exchange.name = 'NYSE' limit 1 ;
Query OK, 1 row affected (0.00 sec)
Records: 1 Duplicates: 0 Warnings: 0

```

Note: We used the two forms which MySQL uses to dis-ambiguate keywords in the preceding MySQL INSERT statement: `Symbol.desc` and the `as 'desc'` forms. Additionally we needed to use the singlequote around the free-standing `'desc'`, which is also a MySQL keyword.

For testing purposes (and because we have fore-knowledge that we are going to need to look up the `SYMBOL.UID` on the row we just added), we then turn around and view the result:

```

mysql> select * from Symbol where Symbol.name like '%NYSE%';
+-----+-----+-----+-----+-----+-----+-----+
| uid  | tid  |  exch  | name          | desc          | conid  |
+-----+-----+-----+-----+-----+-----+-----+
|   11 |   4  |    2   | TRIN-NYSE    | ARMS(TRADING ) |    NULL |
|   12 |   4  |    2   | TICK-NYSE    | ADVANCE - DECLINE |    NULL |
| 5588 |   4  |    2   | AD-NYSE      | NYSE ADVANCE DECLINE INDEX | 33887584 |
+-----+-----+-----+-----+-----+-----+-----+

```

We save for a discussion elsewhere what the proper strategy for populating `SYMBOL.CONID` might be.

Extending the CONTRACT table - part 2

We have covered this topic, with an abbreviated form at [8.1.4](#) 'Extending the CONTRACT table - part 1' and now do so again, but in a less *ad hoc* fashion. A completely formal approach is at `FIXME refformal-load`.

Recall that we had a similar index: `IND.SMART.TICK-NYSE` in the `CONTRACT` table already:

```

mysql> select * from Contract where sid = '12' ;
+-----+-----+-----+-----+-----+
| uid  | sid  | route | unit | tag |
+-----+-----+-----+-----+-----+
| 172  | 12  | 18    | 1    | 0    |
+-----+-----+-----+-----+-----+

```

Only one field changes: the `CONTRACT.SID` to reference the new `Symbol` just added. We view it:

```
mysql> select Symbol.uid as sid, Contract.route as route,
           Contract.unit as unit, Contract.tag as tag from Symbol
           left join Contract on Contract.sid = '12'
           where Symbol.name = 'AD-NYSE' limit 1;
```

```
+-----+-----+-----+-----+
| sid  | route | unit | tag  |
+-----+-----+-----+-----+
| 5588 |    18 |    1 |    0 |
+-----+-----+-----+-----+
```

and add it. Again, we are in the code reference user account:

```
mysql> insert into Contract (sid, route, unit, tag)
       select Symbol.uid as sid, Contract.route as route,
           Contract.unit as unit, Contract.tag as tag from Symbol
           left join Contract on Contract.sid = '12'
           where Symbol.name = 'AD-NYSE' limit 1;
```

```
Query OK, 1 row affected (0.02 sec)
Records: 1 Duplicates: 0 Warnings: 0
```

and of course, we can reverse the lookup, to show that it is correct. we need to add a MySQL ORDER BY clause to force the last row displayed to be the UID just assigned:

```
mysql> select * from Contract order by Contract.uid ;
```

```
+-----+-----+-----+-----+-----+
| uid  | sid  | route | unit | tag  |
+-----+-----+-----+-----+-----+
|    1 |    1 |    9  |    1 |    0 |
| ... |
| 206 | 5588 |    18 |    1 |    0 |
+-----+-----+-----+-----+-----+
```

```
mysql> select Contract.sid, SecType.type, Exchange.name,
           Symbol.name from Contract
           left join Symbol    on Contract.sid  = Symbol.uid
           left join SecType   on Symbol.tid   = SecType.uid
           left join Exchange  on Contract.route = Exchange.uid
           where Contract.uid = '206' ;
```

```
+-----+-----+-----+-----+-----+
| sid  | type | name  | name  |
+-----+-----+-----+-----+-----+
| 5588 | IND  | SMART | AD-NYSE |
+-----+-----+-----+-----+-----+
```

And now we have added a `CONTRACT.UID` for the first symbol mentioned in that IRC thread, that we needed to send the `tick` or `past` commands through the shim. We would repeat the process for `VOL-TICK`, of course, and could then run the queries needed to answer the question asked by the IRC participant.

Conclusion on populating the `SYMBOL` table manually

This concludes our somewhat tactical discussion, about the methods for adding to the various tables by manual efforts. We cover making additions to the bulk load which ultimately can appear in the `CONTRACT` table at `FIXME` rebulk-up.

In that discussion, we will review parts of the MySQL command script: `sql/load.sql`. That is an interesting script, for it has a *'comma' JOIN*, also called an `INNER JOIN`:

```
insert
  into Underlying(nid, home, name, 'desc')
select      6,
           Currency.floor,
           Currency.code,
           Currency.plural
  from Currency, Exchange
  where Currency.floor = Exchange.uid
order by Currency.uid;
```

Note: the fragment: `from Currency, Exchange` which is the *'comma' JOIN*. This produces first the full 'Cartesian product' `JOIN` between `CURRENCY` and `EXCHANGE`, but then limits the result set with the `WHERE` clause.

```
mysql> select Currency.floor, Currency.code, Currency.plural,
           Exchange.name from Currency INNER JOIN Exchange
           where Currency.floor = Exchange.uid;
```

```
+-----+-----+-----+-----+
| floor | code | plural      | name      |
+-----+-----+-----+-----+
|      9 | USD  | US Dollars  | IDEALPRO  |
|      9 | AUD  | AU Dollars  | IDEALPRO  |
|      9 | CAD  | CA Dollars  | IDEALPRO  |
|      9 | CHF  | CH Francs   | IDEALPRO  |
|      9 | EUR  | EU Euro     | IDEALPRO  |
|      9 | GBP  | GB Pounds   | IDEALPRO  |
|      9 | HKD  | HK Dollars  | IDEALPRO  |
```

	9		JPY		JP Yen		IDEALPRO	
	9		MXN		MX Pesos		IDEALPRO	
	9		SEK		SE Kronor		IDEALPRO	
+-----+-----+-----+-----+								

FIXME: subsection

8.1.8 Bulk loading the CONTRACT table

FIXME: possibly move the discussion in tables.tex here

Chapter 9

Commands and the database together

9.1 Market Data, History, and Market Depth

The shim is a command-line and dbms controlled interface; and we have considered some simple database operations in the previous section.

As we now have a small foundation of knowledge for basic database manipulation, as to how to specify a particular contract in which we are interested, we can build a more interesting result than simply looking up and maintaining the shim database. An often requested case is to set up the retrieval of OHLC History data. We use that nomenclature, to distinguish this operation from the more ephemeral retrieval and logging of tick by tick Market Data, which is a form of streaming data.

As an analogy to point up the difference between Market Data and History, consider the ‘play by play’ call by a baseball game radio announcement team, contrasted with the characterization and clerical notation of a box score prepared by the League’s official scorer, each watching the same game.

The first remarks on interesting events as they happen; then the announcer largely ‘forgets’ the minor events in the game’s past once the next potentially interesting event occurs. There may also be a ‘color’ commentator in the booth with the play by play announcer, who may be more versed in baseball trivia and lore. That person might ‘jump in’ from time to time, and offer some context to make the game more interesting for the audience to listen to [compare back in a market context: hitting a new 52 week high, a ‘gap’, or a trading halt].

We mention this as there is both a ‘Market Scanner’ function in the IB TWS API, for which we may add support in the future (as is the customary notation: “... assuming we pick up a committed tester or two for a given new feature, we are certainly willing to discuss adding this to our development”), and it makes sense to build it into this analogy. Also and currently supported in the shim is a facility to run (with a Unix `exec`, which may in turn spawn off a long-lived Unix `fork` process) an external program as part of a `PASTFILTER.SCRIPT`. This permits the ‘downstream client’ of the shim to build in arbitrarily complex add-on processing.

Then there is the official scorer, who summarizes events, possibly ignoring some detail, but (hopefully) memorializing the material highlights of the contest; the scorer may tally and maintain compiled ‘state’ detail such as number of innings pitched or number of strike-out’s by a pitcher, or generate material, such as home run’s hit for very long duration career statistics [we write this the day after Barry Bonds has exceeded Hank Aaron’s record], in a form which might be further summarized for tomorrow’s local newspaper, or for some future edition of *The Bill James’ Baseball Abstract*. These correspond to the Market Data stream, vs. the OHLC History record.

As is our model in this reference, we strive to be practical and task ori-

ented; we pursue at least a couple of objectives in the context of the shim and using the database of the shim. The exercises will build on one after the other.

9.2 Market Data subscription

The retrieval of Market Data, which is commonly called tick data, is handled by managing a ‘subscription’ to the data stream. IB imposes a limit of 100 simultaneous Market Data subscriptions, which limit may be altered upward, depending on commission volume.

The TICKCONFIG table is referenced in the prototype for the tick command, but I do not understand why FIXME. This looks like Market Depth information FIXME.

```
mysql> select * from TickConfig ;
+-----+-----+-----+-----+
| uid | type | bar | bars |
+-----+-----+-----+-----+
|  1 | tick |  5 |   9 |
|  2 | time |  2 |   9 |
|  3 | tick |  5 |  20 |
|  4 | time |  2 |  20 |
+-----+-----+-----+-----+
```

9.2.1 Subscribing to Market Data

The tick command (See: tick [3.24](#)) is a simple command case.

Entering a new subscription consists of:

1. doing the Symbol lookup to determine the CONTRACT.UID (See: Looking up a CONTRACT.UID [8.1.3](#))
2. sending the command:
tick add (Contract.uid) 1;

This assumes one is not close to the limitation on Market Data streams, which IB has in effect on a given account. From observation of the TWS and statements from IB over time, at the time of the 100th simultaneous active request for another Market Data, the TWS will return a message of form:

TBD: example

We are not aware of any way within the API to ‘ask’ what limits are then in effect, and so those constants are hard-coded into the code. If your account has an ability for more, the constant in `src/bind.c` will need to be adjusted and a new shim compiled.

If one goes over the limit, Market Data subscriptions are either: FIXME silently dropped with no further FIXME silently displaced on a FIFO basis and no further FIXME cause an error with XXX consequences.

9.2.2 Unsubscribing from Market Data

The `tick` command (See: [tick 3.24](#)) is a simple command case.

Removing an existing subscription consists of:

1. doing the Symbol lookup to determine the `Contract.uid` (See: [Looking up a CONTRACT.UID 8.1.3](#))
2. sending the command:
`tick del (Contract.uid) 1;`

While the TWS may FIXME does (sample) issue an error message if one attempts to remove a non-subscribed `Contract.uid`, it does not terminate a running TWS session. As the TWS code is obscured from being readable by us, we cannot state with certainty whether any other instability effects may remain behind.

We infer from observation that the last value is ignored in the `del` case of the `tick` command.

9.3 History retrieval

The canonical database walk which we did for `Contract.uid` lookup applies here as well. As we did it the long way earlier, we will provide a couple of summarized recaps here:

The second sample script does a non-recurrent command line history request which is also late enough (070806) that it is calling the `exec-ed` visualization script, `hql2ps`, which reads thus:

```
[herrold@centos-4 bin]$ pwd ; grep past *
/home/herrold/shim/shim-070806/bin
includes: hit_shim 'YM history query' 'past add 179 11;'
includes: ./shim --leaf part tick past
periodic: hit_shim '' 'past add 179 11;';
; sleep60
[herrold@centos-4 bin]$
```

(We have removed a terminal comment, and cleaned up the whitespace in the result above)

```
mysql> select * from PastFilter where PastFilter.uid = '11';
+-----+-----+-----+-----+-----+-----+
| uid | tid | period | reps | duration | end | script |
+-----+-----+-----+-----+-----+-----+
| 11 | 2 | 3 | 0 | 30 | | hql2ps |
+-----+-----+-----+-----+-----+-----+-----+
```

```
mysql> select * from HistoryTag where HistoryTag.uid = '2';
+-----+-----+-----+-----+-----+
| uid | bar | what | format | rth_only |
+-----+-----+-----+-----+-----+
| 2 | 2 | TRADES | ymdt | 1 |
+-----+-----+-----+-----+-----+
```

```
mysql> select * from BarSize where BarSize.uid = '2';
+-----+-----+-----+
| uid | type | secs |
+-----+-----+-----+
| 2 | s05 | 5 |
+-----+-----+-----+
```

Unifying this yields a acutely large query result, which will not fit the printed page. We show the `mysql` query at first, but not its output; then we re-do it in a fashion to show the left, and then the right half of the result, retaining the `PASTFILTER.UID` as a common field:

```
mysql> select PastFilter.uid, PastFilter.period, PastFilter.reps,
             PastFilter.duration, PastFilter.end, PastFilter.script,
             HistoryTag.what, HistoryTag.format, HistoryTag.rth_only,
             BarSize.type, BarSize.secs from PastFilter
left join HistoryTag on HistoryTag.uid = PastFilter.tid
left join BarSize on BarSize.uid = HistoryTag.bar
where PastFilter.uid = '11';
```

```
mysql> select PastFilter.uid, PastFilter.period, PastFilter.reps,
             PastFilter.duration, PastFilter.end from PastFilter
left join HistoryTag on HistoryTag.uid = PastFilter.tid
left join BarSize on BarSize.uid = HistoryTag.bar
where PastFilter.uid = '11';
```

```
+-----+-----+-----+-----+-----+
173
```

```

| uid | period | reps | duration | end |
+-----+-----+-----+-----+-----+
| 11 | 3 | 0 | 30 | |
+-----+-----+-----+-----+

```

```

mysql> select PastFilter.uid, PastFilter.script, HistoryTag.what,
             HistoryTag.format, HistoryTag.rth_only, BarSize.type,
             BarSize.secs from PastFilter
       left join HistoryTag on HistoryTag.uid = PastFilter.tid
       left join BarSize on BarSize.uid = HistoryTag.bar
       where PastFilter.uid = '11';

```

```

+-----+-----+-----+-----+-----+-----+-----+
| uid | script | what  | format | rth_only | type | secs |
+-----+-----+-----+-----+-----+-----+
| 11 | hql2ps | TRADES | ymdt  | 1 | s05 | 5 |
+-----+-----+-----+-----+-----+

```

FIXME This is a example which is running the new scripting facility. Explain this as it also relates to the script helper

9.3.1 Retrieving a History set – one off current

The `past` command (See: [past 3.19](#)) is the simplest case - discuss PASTFILTER options partially

The `shim` attends to doing the database insert of History data retrieved, into the HISTORYBAR table. It also echoed the retrieved data into the logger, and appends a ‘completion’ summarization of what is has added to the database in a message of this type:

```

Aug 7 10:06:46 centos-4 : 6421|36404| 4594668|4|100| 5|
# |4|100|5|event: history insert|(179, 2, 20070807 10:06
:15 -- 20070807 10:06:43)|

```

```

mysql> describe PastFilter ;

```

```

+-----+-----+-----+-----+-----+-----+-----+
| Field      | Type                | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| uid        | int(10) unsigned   | NO   | PRI | NULL    | auto_increment |
| tid        | int(10) unsigned   | NO   | MUL |         |                |
| period     | int(10) unsigned   | YES  | MUL | NULL    |                |
| reps       | int(10) unsigned   | NO   |     | 0       |                |
| duration   | int(10) unsigned   | NO   |     |         |                |
| end        | char(17)           | NO   |     |         |                |

```

script	char(64)	NO				
--------	----------	----	--	--	--	--

We have gone through the individual fields in the discussion of the `past` command previously.

TBD: add a back index link

9.3.2 Retrieving a History set – recurring current

The `past` command (See: [past 3.19](#)) is the next case – discuss `PASTFILTER`, non-null repetition interval

The particular use case we are interested in arises from this message series in the shim’s output logging:

TBD: sample needed here

We use a trimmed down version of the table description of `HISTORYTAG`, as we wish to focus on the the last three fields, with `enum`, `rth`, and `format` field values.

```
mysql> describe HistoryTag ;
+-----+-----+-----+-----+ ...
| Field      | Type                                     |
+-----+-----+-----+-----+
| uid        | int(10) unsigned                       |
| bar        | int(10) unsigned                       |
| what       | enum('TRADES', 'MIDPOINT', 'BID', 'ASK', 'BID/ASK') |
| format     | enum('ymdt', 'epoch')                 |
| rth_only   | tinyint(1)                             |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

9.3.3 History Pacing Violations

The shim has ‘governor logic’ to ‘ration’ the rate at which `past` History requests are sent to the upstream TWS. From observation of the TWS and statements from IB over time, at the time of the 60th request for historical data, the TWS will return a message of form:

```
Oct 10 12:48:32 centos-4 : 8101|46112| 69289766|1|19| 0
|past add 15 13 Ymd_T;|
Oct 10 12:48:32 centos-4 : 8101|46112| 69289799|3| 4| 2
|          15| 162|Historical Market Data Service error
message:Historical data_request pacing violation|
```

```
Oct 10 12:48:32 centos-4 : 8101|46112| 69289955|1| 2| 0
    |ping harvest HMS: 11:00:00 asked at: 20071010 12:48:27
    count: 60 ;|
```

The quick answer on how to avoid this is: Don't do that.

We had thought the shim might again start to receive History data after the expiration of a ten minute interval (like some kind of a period in the 'penalty box' at a hockey game), based on some discussion by others in some of the mailing lists, but we do not observe that with a shim version from late September 2007. After the messages quoted above, the only later message we see is:

```
Oct 10 12:55:01 centos-4 : 8101|46501| 458092368|3| 4| 2
    |      -1|2107|HMDS data farm connection is inactive
    but should be available upon demand.:ushmds2a|
```

and nothing more, after waiting for another fifteen minutes beyond that.

However, this governor logic is configurable, so that one can manage the rate limiting in a downstream client, rather than being constrained by the shim. TBD: explain the RC file option

We produced the error message above by altering the governor logic limits temporarily to permit one per second past queries, and feeding the shim thus:

```
[herrold@centos-4 shim.070928]$ ../get_day.sh | ./shim --data logd
```

with the following script, which we have also discussed in other forms on the mailing list:

```
[herrold@centos-4 shim]$ cat get_day.sh
#!/bin/sh
#
#      emit a series of commands to get a day's worth of History
#      on Contract.uid 15 (AIG), using PastFilter.uid 13
#
CID="15"
QRY="13"
DOZE="1"
N="0"
#
#####3
#
for k in `seq 10 24`; do
#
```

```

#       only weekdays
      [ 'date --date "200709${k}" +%u ' -lt 6 ] && {
#
#       brand when we ask
      BRAND='date +%Y%m%m %T' '
      echo "ping harvest day: 200709${k} asked at: ${BRAND} ;"
for i in seq 9 16; do
      [ $i -gt 9 ] && {
      BRAND='date +%Y%m%m %T' '
      export N='echo "${N} + 1" | bc'
      echo "ping harvest HMS: ${i}:00:00 asked at: ${BRAND} count: ${N} ;"
          echo -n "past add ${CID} ${QRY} Ymd_T(200709"
          echo -n "${k}"
          echo -n " "
          echo "${i}:00:00);"
#           echo "wait 10;"
      sleep ${DOZE}
          }
      [ $i -lt 16 ] && {
      BRAND='date +%Y%m%m %T' '
      export N='echo "${N} + 1" | bc'
      echo "ping harvest HMS: ${i}:00:00 asked at: ${BRAND} count: ${N} ;"
          echo -n "past add ${CID} ${QRY} Ymd_T(200709"
          echo -n "${k}"
          echo -n " "
          [ $i -lt 10 ] && echo -n "0"
          echo "${i}:30:00);"
#           echo "wait 10;"
      sleep ${DOZE}
          }
done
    }
done
#
# echo "quit;";
#

```

As a bit of good news, we note that by varying the 'DOZE' parameter back to 11 seconds, in that same series of tests we were able run the 154 past commands in succession, This permitted us to harvest a bit over a quarter million lines of 'per second' AIG History detail without incident in about one-half hour's elapsed time (a bit less actually: 1694 seconds):

```
mysql> select count(*) from HistoryBar;
+-----+
| count(*) |
+-----+
| 280800 |
+-----+
```

9.4 Market Depth subscription

The retrieval of Market Depth, which is commonly called ‘OpenBook’ data, is handled by managing a ‘subscription’ to the data stream IB imposes a limit of 3 simultaneous Market Depth subscriptions, which limit may be altered upward, depending on commission volume.

FIXME – more text here

The DEPTHLIMIT table is referenced in the prototype for the `book` command. It permits a lookup of the desired number of Market Depth (so called ‘OpenBook’) visible orders.

```
mysql> select * from DepthLimit ;
+-----+-----+
| uid | rows |
+-----+-----+
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | 10 |
+-----+-----+
```

In the usual case, we caution about the need to not rely on the strict sequential relation between the `uid` and the datum referenced, and to counsel doing the database lookup. In this particular case, however, this should be a durable relation.

9.4.1 Subscribing to Market Depth

The `book` command (See: [book 3.4](#)) is the simplest case

1. doing the Symbol lookup to determine the `CONTRACT.UID` (See: Looking up a `CONTRACT.UID` [8.1.3](#))
2. sending the command:
`book add (Contract.uid) 9;`

This assumes one is not close to the limitation which IB has in effect on a given account. We are aware of no way within the API to ‘ask’ what limits are in effect, and so those constants are presently hard-coded into the code.

If one goes over the limit, Market Depth subscriptions are either:

FIXME silently dropped with no further

FIXME silently displaced on a FIFO basis and no further

FIXME cause an error with XXX consequences.

9.4.2 Unsubscribing from Market Depth

The `book` command (See: [book 3.4](#)) is a simple command case:

1. doing the Symbol lookup to determine the `CONTRACT.UID` (See: Looking up a `CONTRACT.UID` [8.1.3](#))
2. sending the command:
`book del (Contract.uid) 1;`

While the TWS may FIXME does (sample) issue an error message if one attempts to remove a non-subscribed `CONTRACT.UID`, it does not terminate a running TWS session. As the TWS code is obscured from being readable by us, we cannot state with certainty whether any other instability effects may remain behind.

We infer from observation that the last value is ignored in the `del` case of the `book` command.

Chapter 10

Adding a web browser interface

10.1 Look up interface

TBD - show a lookup interface – script luCID.sh is an example

Testing – Add code to show the screen-shot interface I use:

```
#!/bin/sh
[ "x$1" = "x" ] && {
    echo "usage: $0 filename.jpg"
}
cd ~
sleep 10
import -window root $1
```

We discuss diagnosis, and point to solutions for the problems we have faced along the way.

Part IV

Preparing this document

Chapter 11

Preparing this document

How we generate this document – a transient part until it stabilizes

Chapter 12

The writing process

12.1 Adding new commands

1. periodically inventory new commands (YYMMDD will of course vary)

```
( ./get_cmd_list.sh ../shim-070706 | \  
grep cmd ; ls -1 *tex ) > command-list-070709.txt
```

and print it

2. when a new command appears, copy `template.tex` to `command-name.tex`
3. add a placeholder

```
% FIXME \include(command-name.tex)
```

in the alphabetical command list in

```
commands.tex
```

with the FIXME marker

4. edit the new

```
command-name.tex
```

to contain a marker

```
\section{command-name - FIXME}
```

5. update `shim-help.tex`, top section

12.2 Editing prior text

1. spot remaining items to edit thus:

```
grep FIXME *tex | grep -v ':%'
```

and process newly appearing commands in, as they start working

2. clean up older commands containing ^FIXME's, ^TBD's and ^QUERY's

Run an 'edit' window, and a 'make' window

```
make clean ; make all && xdvi commands.dvi
```

Part V
Conclusion

This ends the document – an ‘under construction’ part until we draw more substantive conclusions

“FEED ME, NORMAN ... FEED ME”

FIXME - Appendix template

Bibliography

[trading-shim home page] The trading-shim home page
<http://www.trading-shim.org/>

[trading-shim manual] The trading-shim manual
<http://www.trading-shim.org/pdfs/manual.pdf>

[Russ' command reference - DRAFT] Russ' command reference
<http://www.herrold.com/commands.pdf>

Index

- [.shimrc](#), [108](#)
- [account](#), [22](#)
- [account](#), [22](#)
- [acct](#), [23](#)
 - [help](#), [26](#)
- [Appendix](#), [195](#)
- [args](#), [101](#)
 - [help](#), [101](#)
- [Arguments](#), [101](#)
- [Bibliography](#), [198](#)
- [bind](#), [27](#)
- [book](#), [28](#)
 - [help](#), [30](#)
- [cash](#), [31](#)
- [cmds](#), [102](#)
 - [help](#), [102](#)
- [Code](#)
 - [../snapshot.sh](#), [182](#)
- [Commands](#), [102](#)
 - [account](#), [22](#)
 - [acct](#), [23](#)
 - [bind](#), [27](#)
 - [book](#), [28](#)
 - [cash](#), [31](#)
 - [dbms](#), [32](#)
 - [exec](#), [34](#)
 - [exercise](#), [35](#)
 - [feed](#), [36](#)
 - [help](#), [38](#), [100](#)
 - [history](#), [40](#)
 - [info](#), [41](#)
 - [list](#), [43](#)
 - [load](#), [45](#)
 - [news](#), [48](#)
 - [next](#), [50](#)
 - [open](#), [52](#)
 - [order](#), [54](#)
 - [past](#), [55](#)
 - [ping](#), [72](#)
 - [quit](#), [74](#)
 - [read](#), [76](#)
 - [scan](#), [78](#)
 - [Syntax](#), [15](#)
 - [tick](#), [79](#)
 - [transmit](#), [83](#)
 - [verb](#), [84](#)
 - [wait](#), [86](#)
 - [wake](#), [88](#)
 - [wild](#), [91](#)
 - [wire](#), [92](#)
 - [xmit](#), [94](#)
- [Conclusion](#), [193](#)
- [CONTRACT](#)
 - [Bulk loading](#), [168](#)
- [Copyright](#), [iii](#)
- [Copyrighted material](#)
 - [Quotation](#), [4](#)
- [Cover Page](#), [1](#)
- [Currency](#)
 - [Looking up from Type, Route, and Symbol](#)
 - [directly with LEFT JOIN](#), [146](#)
- [Currency \(Forex\)](#)
 - [Looking up from a CONTRACT.UID](#)
 - [directly with LEFT JOIN](#), [144](#)
- [Currency Unit](#)
 - [Looking up from a CONTRACT.UID](#), [139](#)

- dbms, [32](#)
- Disclaimer, [xiii](#)
- error message
 - can't parse long string, [56](#)
 - Historical data query end date/time string is invalid, [56](#)
 - Historical Market Data Service error message, [175](#)
 - No security definition has been found for the request, [137](#)
- exec, [34](#)
- exercise, [35](#)
- Extending the CONTRACT table
 - INSERT
 - part 1, [150](#)
 - part 2, [164](#)
- feed, [36](#)
- Files, [108](#)
 - .shimrc, [108](#)
 - help, [108](#)
 - ShimText, [111](#)
- Foreign Exchange
 - see: Currency, [144](#)
- Forex
 - see: Currency, [25](#), [144](#)
- Future
 - Expiration
 - Looking up from a FUTDETAIL.UID, [140](#)
 - Looking up from a CONTRACT.UID
 - directly with LEFT JOIN, [142](#)
 - Looking up from Type, Route, and Symbol
 - directly with LEFT JOIN, [147](#)
 - Security Type
 - Looking up from a CONTRACT.TAG, [139](#)
- governor logic
 - History, [175](#)
- help, [38](#), [100](#)
- .shimrc, [108](#)
- acct, [26](#)
- args, [101](#)
- book, [30](#)
- cmds, [102](#)
- help, [39](#), [100](#)
- info, [42](#)
- link, [33](#), [37](#), [103](#)
- list, [44](#)
- load, [47](#)
- mode, [104](#), [105](#)
- news, [49](#)
- next, [51](#)
- open, [53](#)
- opts, [106](#), [107](#)
- past, [71](#)
- ping, [73](#)
- quit, [75](#)
- read, [77](#)
- tick, [82](#)
- verb, [85](#)
- wait, [87](#)
- wake, [90](#)
- wire, [93](#)
- History
 - governor logic, [175](#)
 - original, [63](#)
 - Pacing Violations, [175](#)
 - rate limit, [175](#)
 - Retrieving
 - one off current, [174](#)
 - recurring current, [175](#)
 - smoothed, [63](#)
- history, [40](#)
- Home Exchange
 - Looking up from an EXCHANGE.UID, [138](#)
- Index, [198](#)
- Index symbol
 - Looking up from a CONTRACT.UID directly with LEFT JOIN, [143](#)

- Looking up from Type, Route, and Symbol
 - directly with LEFT JOIN, [147](#)
- info, [41](#)
 - help, [42](#)
- Introduction, [3](#)
- IRC
 - irc.ether.net, [158](#)
 - #interactivebrokers, [158](#)
- License, [xv](#)
 - GPL v 3, [xv](#)
- link, [103](#)
 - help, [33](#), [37](#), [103](#)
- Links, [103](#)
- list, [43](#)
 - help, [44](#)
- load, [45](#)
 - help, [47](#)
- logd, [111](#)
- make test
 - Fixing an error, [151](#)
- Market Data
 - Subscribing, [171](#)
 - Unsubscribing, [172](#)
- Market Data Subscription, [171](#)
- Market Depth
 - Subscribing, [178](#)
 - Unsubscribing, [179](#)
- Market Depth Subscription, [178](#)
- mode, [104](#)
 - help, [104](#), [105](#)
- Modes, [104](#)
- MySQL
 - 'comma' JOIN, [166](#)
 - account rights
 - per userid, [149](#)
 - alias operator
 - AS, [164](#)
 - backup client
 - mysqldump, [132](#)
 - command line client
 - mysql, [128](#), [137](#)
 - mysqldump, [132](#)
 - DELETE FROM, [67](#)
 - DESCRIBE, [57](#)
 - foreign keys, [129](#), [132](#), [149](#)
 - GROUP BY, [63](#)
 - INCLUDE, [65](#)
 - INNER JOIN, [166](#)
 - INSERT, [59](#), [63](#), [132](#), [136](#), [149](#), [162](#)
 - JOIN, [166](#)
 - LEFT JOIN, [60](#), [64](#), [136](#), [142](#)
 - match operator
 - LIKE, [144](#)
 - ORDER BY, [63](#), [136](#), [165](#)
 - query client
 - mysql, [128](#), [137](#)
 - SELECT, [63](#), [136](#), [142](#)
 - Status, [163](#)
 - WHERE, [136](#), [166](#)
 - wildcard match character
 - %, [144](#)
- news, [48](#)
 - help, [49](#)
- next, [50](#)
 - help, [51](#)
- numbering, [110](#)
 - Commands
 - rule.c, [120](#)
 - Comments
 - rule.c, [122](#)
 - Java sample client, [117](#)
 - Message, [118](#)
 - Request, [117](#)
 - tick types, [119](#)
 - Message
 - Java sample client, [118](#)
 - Messages
 - rule.c, [122](#)
 - Request
 - Java sample client, [117](#)
 - rule.c, [121](#)

- rule.c, [120](#)
 - Commands, [120](#)
 - Comments, [122](#)
 - Messages, [122](#)
 - Request, [121](#)
- tick types
 - Java sample client, [119](#)
- open, [52](#)
 - help, [53](#)
- OpenBook, [178](#)
- Options
 - see: shim
 - shim options, [106](#)
- opts, [106](#)
 - help, [106](#), [107](#)
- order, [54](#)
- past, [55](#)
 - help, [71](#)
 - PASTFILTER look up, [60](#)
- ping, [72](#)
 - help, [73](#)
- Preparing this document, [187](#)
- quit, [74](#)
 - help, [75](#)
- rate limit
 - History, [175](#)
- read, [76](#)
 - help, [77](#)
- Route
 - Looking up from an EXCHANGE.UID, [138](#)
- scan, [78](#)
- shim
 - Arguments, [101](#)
 - Commands, [102](#)
 - Files, [108](#)
 - ShimText, [111](#)
 - Links, [103](#)
 - Modes, [104](#)
 - modes, [104](#)
 - data, [104](#)
 - help, [104](#)
 - play, [104](#)
 - risk, [104](#)
 - unit, [104](#)
 - shim options, [106](#)
 - cout, [106](#)
 - fast, [106](#)
 - file, [106](#)
 - init, [106](#)
 - load, [106](#)
 - logd, [106](#), [111](#)
 - pane, [106](#)
 - save, [106](#)
 - table
 - CONTRACT, [29](#), [41](#), [55](#), [79](#), [132](#)
 - CURRENCY, [132](#)
 - LINEITEM, [76](#)
 - LOCALSET, [129](#), [132](#)
 - MISCELLANY, [132](#)
 - PRODUCTMAP, [132](#)
 - STOCK, [132](#)
 - SUBREQUEST, [45](#), [129](#)
 - SYMBOL, [132](#)
 - TICKCONFIG, [80](#)
 - UNDERLYING, [132](#)
 - WATCHSETS, [129–131](#)
 - shim options
 - logd, [111](#)
 - ShimText, [111](#)
 - Stock
 - Looking up from a CONTRACT.UID
 - directly with LEFT JOIN, [143](#)
 - Looking up from Type, Route, and Symbol
 - directly with LEFT JOIN, [146](#)
 - subscription limit
 - Market Data, [171](#)
 - Market Depth, [178](#)
 - tickstream, [171](#)
 - Symbol

Adding to the CONTRACT table, help, 85
147
Extending the SYMBOL table, 158 wait, 86
automation, 162 help, 87
INSERT, 162 wake, 88
manually, 159 help, 90
Looking up from a CONTRACT.UID Warranties disclaimer, 4
directly with LEFT JOIN, 142 wild, 91
step by step, 136 wire, 92
Looking up from a SYMBOL.UID, help, 93
138 xmit, 94

Table of Contents, xi
Tables, 134
tick, 79
help, 82
tick data, 171
To Do, xv
Trademarks, 3
transmit, 83
Troubleshooting, 184
Tutorial, 126
Typographic conventions, 4

Unix
cat, 116
cut, 112
exec, 170
find, 116
firefox, 115
fork, 170
grep, 120, 137
head, 112
jar, 116
less, 117
lpr, 117
ls, 116
sed, 151
stderr, 38, 109
stdin, 109
stdout, 38

verb, 84

