#!/dev/cartel

TclRFA 7.5

Reference Guide

11 February 2014 | www.devcartel.com

CONTENTS

ABOUT TCLRFA

TclRFA is built over Thomson Reuters' high performance RFA C++ API to be used in Tcl programming language. But why Tcl, you might ask? The first reason is that Tcl is probably the most kept secret scripting language in the entire industry with powerful regular expression engine, flexible syntax, package management and ease of building a C extension. The second reason lies in its nature of dynamic typing as opposed to static typing in C++ and Java that makes coding less complicated. We hide abstract layers of RFA API, take care of redundant function calls in C, internally encode/decode messages and expose the simplest interfaces to a user and yet maintain the strict concept of RFA such as Session, Config Database, EventQueue.

With TclRFA, you can have a working Thomson Reuters market data consumer application in an amazingly simple 20 lines of code (see below sample code) and not thousands as in traditional C++. Better yet, the same script runs on both Windows and Linux without modifying of base scripts.

FEATURES

- Subscription for MARKET_PRICE (level 1)
- Subscription for MARKET_BY_ORDER (order book)
- Subscription for MARKET_BY_PRICE (market depth)
- Snapshot request (no incremental updates)
- Dictionary download or use local files
- Directory request
- Symbol list request
- Time series request and decoder for IDN TS1
- Custom domain MMT_HISTORY which can be used for intraday time-series publishing
- Non-interactive provider for MARKET_PRICE, MARKET_BY_ORDER, MARKET_BY_PRICE, SYMBOLLIST, HISTORY
- Debug mode
- Logging
- Low-latency mode
- Subscription outbound NIC binding
- Example scripts

GETTING STARTED

To be familiar with TclRFA, please follow the steps below:

1. Download the latest TclRFA from www.devcartel.com/tclrfa.
2. Extract the package on your workstation.
3. Open the `/example` folder
4. Open and edit `tclrfa.cfg` for `provider01.tcl` and `consumer01.tcl` as follows:

```
#Consumer

\Connections\Connection_RSSL1\rsslPort = "14002"

\Connections\Connection_RSSL1\ServerList = "<P2PS or ADS ip>"

...

#Provider

\Connections\Connection_RSSL4\rsslPort = "14003"

\Connections\Connection_RSSL4\ServerList = "<MDH or ADH ip>"
```

5. Start `/example/provider.tcl` with the command below, wait for the data to be published to MDH/ADH cache:

```
$ tclsh provider.tcl
```

6. Start `/example/consumer01.tcl` to consume data while running `provider.tcl` with:

```
$ tclsh consumer01.tcl
```

7. `consumer01.tcl` retrieves a market data full image followed by incremental updates of 'EUR=', below is the execution result:

```
[Mon Jul 15 14:37:40 2013]: (ComponentName) Tclrfa: (Severity) Information:
[Tclrfa::login] Login successful. (username: tclrfa)

[DictionaryHandler::DictionaryHandler] Successfully load dictionaries from
../etc/RDM/RDMFieldDictionary, ../etc/RDM/enumtype.def

{NIP EUR= REFRESH} {NIP EUR= { RDNDISPLAY {100} RDN_EXCHID {SES} BID {0.988} ASK
{0.999} DIVPAYDATE {23 JUN 2011} }}

{NIP EUR= { BID_NET_CH {0.0041} BID {0.988} ASK {0.999} ASK_TIME {14:37:40:921} }}
```

FUNCTIONS

Initialization

[tclrfa]

Instantiate a TclRFA object

```
% set t [tclrfa]
```

setDebugMode *True|False*

Enable or disable debug messages.

```
% $t setDebugMode True
```

Configuration

createConfigDb *filename*

filename = configuration file location, full or relative path

Read the configuration file. Initialize RFA context.

```
% $t createConfigDb "./tclrfa.cfg"

[Tclrfa::initializeRFA] Initializing RFA context...

[Tclrfa::createConfigDb] Loading RFA config from file: ./tclrfa.cfg
```

printConfigDb *config_node*

config_node = part of the configuration node to be printed out

Print current configuration values. If the arg is omitted, this function returns all of the configuration values under the ‘Default’ namespace.

```
% $t printConfigDb "\\Default\\Sessions"

\Default\Sessions\Session1\connectionList = Connection_RSSL1

\Default\Sessions\Session2\connectionList = Connection_RSSL2

\Default\Sessions\Session3\connectionList = Connection_RSSL3

\Default\Sessions\Session4\connectionList = Connection_RSSL4
```

Session

acquireSession *session*

session = session name

Acquire a session as defined in the configuration file and look up for an appropriate connection type then establish a client/server network session.

```
% $t acquireSession "Session1"
```

Client

createOMMConsumer

Create an OMM data consumer client.

```
% $t createOMMConsumer
```

createOMMProvider

Create an OMM data provider client.

```
% $t createOMMProvider
```

login *?username instanceid appid?*

username = DACS user ID

instanceid = unique application instance ID

appid = application ID (1-256)

Send an authentication message with user information. This step is mandatory in order to consume the market data from P2PS/ADS. If arguments are omitted, TclRFA will read values from configuration file.

```
% $t login

...

[Thu Jul 04 17:56:48 2013]: (ComponentName) Tclrfa: (Severity) Information:
[Tclrfa::login] Login successful. (username: TclRFA)
```

isLoggedIn

Check whether the client successfully receives a login status from the P2PS/ADS.

```
% $t isLoggedIn

1
```

setInteractionType *type*

type = “snapshot” or “streaming”

Set subscription type ‘snapshot’ or ‘streaming’. If ‘snapshot’ is specified, the client will receive only the full image of an instrument then the subscribed stream will be closed.

```
% $t setInteractionType “snapshot”
```

Directory

directoryRequest

Send a directory request through the acquired session. This step is the mandatory in order to consume the market data from P2PS/ADS.

```
% $t directoryRequest
```

submitDirectory *domain*

domain = data domain number

Submit directory with domain type (capability) in a provider application, it currently supports:

- 6 - market price
- 7 - market by order
- 8 - market by price
- 10 - symbol list
- 12 - history

This function is normally called automatically upon data submission.

```
% $t submitDirectory 6
```

Dictionary

dictionaryRequest

If `downloadDataDict` configuration is set to `True` then TclRFA will send a request for data dictionaries to P2PS/ADS. Otherwise, it uses local data dictionaries specified by `fieldDictionaryFilename` and `enumTypeFilename` from configuration file.

```
% $t dictionaryRequest
```

isNetworkDictionaryAvailable

Check whether the data dictionary is successfully downloaded from the server.

```
% $t isNetworkDictionaryAvailable

1
```

Logging

log *message*

Write a normal message to a log file.

```
% $t log “Print log message out”

[Thu Jul 04 17:45:29 2013]: (ComponentName) Tclrfa: (Severity) Information: Print log message out
```

logWarning *message*

Write a warning message to a log file.

```
% $t logWarning “Print warning message out”

[Thu Jul 04 17:47:03 2013]: (ComponentName) Tclrfa: (Severity) Warning: Print warning message out
```

logError *message*

Write an error message to a log file.

```
% $t logError “Print error message out”

[Thu Jul 04 17:48:00 2013]: (ComponentName) Tclrfa: (Severity) Error: Unexpected error: Print error message out
```

Symbol List

symbolListRequest *RIC ?RIC RIC ...?*

RIC = Reuters Instrument Code

For consumer application to subscribe symbol lists. User can define multiple symbol list names. Data dispatched using `dispatchEventQueue` function.

Image

```
{ <SERVICE_NAME> <SYMBOLLIST_NAME> REFRESH }
```

```
{ <SERVICE_NAME> <SYMBOLLIST_NAME> <COMMAND> { ITEM_NAME { <FID_NAME#1> {<VALUE#1>} ... <FID_NAME#X> {<VALUE#X>}} }
```

Update

```
{ <SERVICE_NAME> <SYMBOLLIST_NAME> <COMMAND> { ITEM_NAME { <FID_NAME#1> {<VALUE#1>} ... <FID_NAME#X> {<VALUE#X>}} }
```

```
% $t symbolListRequest "0#BMD"

...

% $t dispatchEventQueue

[SymbolListHandler::processResponse] SymbolList Refresh: 0#BMD.NIP {NIP 0#BMD REFRESH} {NIP 0#BMD ADD {FCPO { PROD_PERM {10} PROV_SYMB {MY439483}}}}
```

symbolListCloseRequest *RIC ?RIC RIC ...?*

RIC = Reuters Instrument Code

Unsubscribe the specified symbol lists. User can define multiple symbol list names

```
% $t symbolListCloseRequest “0#BMD” “0#ARCA”
```

symbolListCloseAllRequest

Unsubscribe all symbol lists in the watch list.

```
% $t symbolListCloseAllRequest
```

isSymbolListRefreshComplete

Check whether the client receives a complete list of the symbol list.

```
% $t isSymbolListRefreshComplete

1
```

getSymbolList *RIC*

RIC = Reuters Instrument Code

Return item names available under a symbol list in string format.

```
% $t getSymbolList “0#BMD”

FPCO FPKC FPRD FPGO
```

getSymbolListWatchList

Return names of the subscribed symbol lists.

```
% $t getSymbolListWatchList

0#BMCA 0#ARCA
```

symbolListSubmit *command data*

command = instruction to manipulate a list

data = Tcl dict

For a provider client to publish a list of symbols to MDH/ADH under data domain 10, available commands are:

- ADD - add a new symbol to the list
- UPDATE - update a symbol’s attributes
- DELETE - delete a symbol from the list

The second one is a Tcl dict in following format:

```
<SYMBOLLIST_NAME> <ITEM#1_NAME> {<FID_NAME#1> {<VALUE#1>} <FID_NAME#2> {<VALUE#2>} ...
<FID_NAME#X> {VALUE#X}}
```

```
dict set SYMBOLLIST 0#BMD FCPO { PROD_PERM {10} PROV_SYMB {MY439483} }

% $t symbolListSubmit add $SYMBOLLIST

[Tclrfa::symbolListSubmit] mapAction: add

[Tclrfa::symbolListSubmit] symbolName: 0#BMD

[Tclrfa::symbolListSubmit] mapKey: FCPO

[Tclrfa::symbolListSubmit] fieldList: [Tclrfa::symbolListSubmit] fieldList:
PROV_SYMB=MY439483,PROD_PERM=10
```

Market Price

marketPriceRequest *RIC ?RIC RIC ...?*

RIC = Reuters Instrument Code

For consumer client to subscribe market data from P2PS/ADS, user can define multiple item names. The data dispatched through `dispatchEventQueue` function in Tcl dict format:

Image

```
{<SERVICE_NAME> <ITEM_NAME> REFRESH} {<SERVICE_NAME> <ITEM_NAME> {<FID_NAME#1>
{<VALUE#1>} <FID_NAME#2> {<VALUE#2>} ... <FID_NAME#3> {<VALUE#X>}}}
```

Update

```
{<SERVICE_NAME> <ITEM_NAME> {<FID_NAME#1> {<VALUE#1>} <FID_NAME#2> {<VALUE#2>} ...
<FID_NAME#3> {<VALUE#X>}}}
```

```
 % $t marketPriceRequest “EUR=”

 % $t dispatchEventQueue 10

{NIP EUR= REFRESH} {NIP EUR= { RDNDISPLAY {100} RDN_EXCHID {SES} BID {0.988} ASK
{0.999} DIVPAYDATE {23 JUN 2011} }}
```

marketPriceCloseRequest *RIC ?RIC RIC ...?*

RIC = Reuters Instrument Code

Unsubscribe items from streaming service. User can define multiple item names.

```
 % $t marketPriceCloseRequest “C.N” “JPY=”
```

marketPriceCloseAllRequest

Unsubscribe all items from streaming service.

```
% $t marketPriceCloseAllRequest
```

getMarketPriceWatchList

Returns names of the subscribed items.

```
% $t getMarketPriceWatchList

C.N JPY=
```

marketPriceSubmit *data*

data = Tcl dict

For provider client to publish market data to MDH/ADH, the market data image/update must be in the following Tcl dict format:

```
<ITEM_NAME> {<FID_NAME#1> {<VALUE#1>} ... <FID_NAME#X> {<VALUE#X>}}
```

```
% dict set IMAGES EUR= { RDNDISPLAY {100} RDN_EXCHID {155} BID {0.988} ASK {0.999}
DIVPAYDATE {20110623} }

% dict set IMAGES C.N { RDNDISPLAY {100} RDN_EXCHID {NAS} OFFCL_CODE {isin1234XYZ}
BID {4.23} DIVPAYDATE {20110623} OPEN_TIME {09:00:01.000} }

% $t marketPriceSubmit $IMAGES

[Tclrfa::marketPriceSubmit] EUR= { RDNDISPLAY {100} RDN_EXCHID {155} BID {0.988} ASK
{0.999} DIVPAYDATE {20110623} } C.N { RDNDISPLAY {100} RDN_EXCHID {NAS} OFFCL_CODE
{isin1234XYZ} BID {4.23} DIVPAYDATE {20110623} OPEN_TIME {09:00:01.000} }

[Tclrfa::marketPriceSubmit] symbolName: EUR=

[Tclrfa::marketPriceSubmit] fieldList:
RDNDISPLAY=100,RDN_EXCHID=155,BID=0.988,ASK=0.999,DIVPAYDATE=20110623,

[Tclrfa::marketPriceSubmit] symbolName: C.N

[Tclrfa::marketPriceSubmit] fieldList:
RDNDISPLAY=100,RDN_EXCHID=NAS,OFFCL_CODE=isin1234XYZ,BID=4.23,DIVPAYDATE=20110623,OPE
N_TIME=09:00:01.000,

[Tclrfa::marketPriceSubmit] EUR= {BID_NET_CH 0.0041 BID 0.988 ASK 0.999 ASK_TIME now}
C.N {ACVOL_1 1001 TRDPRC_1 4.561 TIMACT now}
```

Market by Order

marketByOrderRequest *RIC ?RIC RIC ...?*

RIC = Reuters Instrument Code

For a consumer application to subscribe order book data, user can define multiple item names. The data dispatched through `dispatchEventQueue` in Tcl dict format:

Image

```
{<SERVICE_NAME> <ITEM_NAME> REFRESH} {<SERVICE_NAME> <ITEM_NAME> <COMMAND> { ORDER_ID {
<FID_NAME#1> {<VALUE#1>} ... <FID_NAME#X> {<VALUE#X>}}}
```

Update

```
{<SERVICE_NAME> <ITEM_NAME> <COMMAND> { ORDER_ID { <FID_NAME#1> {<VALUE#1>} ...
<FID_NAME#X> {<VALUE#X>}}}
```

```
 % $t marketByOrderRequest "ANZ.AX"

 % $t dispatchEventQueue 10000

{NIP ANZ.AX REFRESH} {NIP ANZ.AX ADD {538993C200035057B { ORDER_PRC {20.25} ORDER_SIZE
{369} ORDER_SIDE {BID} SEQNUM_QT {804} EX_ORD_TYP {0} CHG_REAS {10} ORDER_TONE {} }}}
{NIP ANZ.AX ADD {538993C200083483B { ORDER_PRC {20.280} ORDER_SIZE {100} ORDER_SIDE
{BID} SEQNUM_QT {2744} EX_ORD_TYP {0} CHG_REAS {6} ORDER_TONE {} }}}
```

marketByOrderCloseRequest *RIC ?RIC RIC ...?*

RIC = Reuters Instrument Code

Unsubscribe items from order book streaming service. User can define multiple item names.

```
% $t marketByOrderCloseRequest "ANZ.AX"
```

marketByOrderCloseAllRequest

Unsubscribe all items from order book data streaming service.

```
% $t marketByOrderCloseAllRequest
```

getMarketByOrderWatchList

Return all subscribed item names on order book streaming service.

```
% $t getMarketByOrderWatchList

 ANZ.AX
```

marketByOrderSubmit *command data*

command = instruction to manipulate an order book

data = Tcl dict

For a provider client to publish specified order book data to MDH/ADH, `marketByOrderSubmit` requires two parameters, the first one is the order book placement command:

- ADD - add new order to the item
- UPDATE - update an order's attributes
- DELETE - remove an order from the book

The second one is a Tcl dict and must be in the following format:

```
<ITEM_NAME> <ORDER_ID> {<FID_NAME#1> {<VALUE#1>} <FID_NAME#2> {<VALUE#2>} ...
<FID_NAME#X> {<VALUE#X>}}
```

```
% dict set ORDERS ANZ.AX 538993C200035057B { ORDER_PRC {20.260} ORDER_SIZE {50}
ORDER_SIDE {BID} SEQNUM_QT {2744} EX_ORD_TYP {0} CHG_REAS {6} ORDER_TONE {} }

% $t marketByOrderSubmit('ADD', ORDERS)

[Tclrfa::dispatchEventQueue] Event loop - approximate pending Events: 0

[Tclrfa::marketByOrderSubmit] mapAction: add

[Tclrfa::marketByOrderSubmit] symbol name: ANZ.AX

[Tclrfa::marketByOrderSubmit] mapKey: 538993C200035057B

[Tclrfa::marketByOrderSubmit] fieldList:
ORDER_PRC=20.260,ORDER_SIZE=50,ORDER_SIDE=BID,SEQNUM_QT=2744,EX_ORD_TYP=0,CHG_REAS=6,OR
DER_TONE=,

[OMMCProvServer::submitData] sending refresh item: ANZ.AX

[OMMCProvServer::submitData] sending refresh service: NIP
```

Market by Price

marketByPriceRequest *RIC ?RIC RIC ...?*

RIC = Reuters Instrument Code

For consumer application to subscribe market depth data, user can define multiple item names. Data dispatched through `dispatchEventQueue` module in a tuple below:

Image

```
{ <SERVICE_NAME> <ITEM_NAME> REFRESH } { <SERVICE_NAME> <ITEM_NAME> <COMMAND> { DEPTH {
<FID_NAME#1> {<VALUE#1>} ... <FID_NAME#X> {<VALUE#X>}}}
```

Update

```
{ <SERVICE_NAME> <ITEM_NAME> <COMMAND> { DEPTH { <FID_NAME#1> {<VALUE#1>} ...
<FID_NAME#X> {<VALUE#X>}}}
```

```
% $t marketByOrderRequest "ANZ.CHA"

% $t dispatchEventQueue 1000

{NIP ANZ.CHA REFRESH} {NIP ANZ.CHA ADD {210000B { ORDER_PRC {21.0000} ORDER_SIDE {ASK}
ORDER_SIZE {500} NO_ORD {13} QUOTIM_MS {16987567} ORDER_TONE {} }}} {NIP ANZ.CHA ADD
{201000B { ORDER_PRC {20.1000} ORDER_SIDE {BID} ORDER_SIZE {97} NO_ORD {2} QUOTIM_MS
{16987567} ORDER_TONE {} }}}
```

marketByPriceCloseRequest *RIC ?RIC RIC ...?*

RIC = Reuters Instrument Code

Unsubscribe items from market depth data stream. User can define multiple item names.

```
% $t marketByOrderCloseRequest "ANZ.AX"
```

marketByPriceCloseAllRequest

Unsubscribe all items from market depth stream.

```
% $t marketByPriceCloseAllRequest
```

getMarketByPriceWatchList

Return all subscribed item names on market depth streaming service.

```
% t getMarketByPriceWatchList

 ANZ.CHA
```

marketByPriceSubmit *command data*

command = instruction to manipulate an order book

data = Tcl dict

For a provider client to publish the specified market depth data to MDH/ADH, `marketByPriceSubmit` requires two parameters, the first one is the depth placement command:

- ADD – add a new price to the depth
- UPDATE - update a price level's attributes
- DELETE - remove a price from the depth

The second one is a Tcl dice and must be in the following format:

```
<ITEM_NAME> <DEPTH> {<FID_NAME#1> {<VALUE#1>} <FID_NAME#2> {<VALUE#2>} ...  <FID_NAME#X> {<VALUE#X>}}
```

```
% dict set DEPTHS ANZ.CHA 201000B { ORDER_PRC {20.1000} ORDER_SIDE {BID} ORDER_SIZE {1300} NO_ORD {13} QUOTIM_MS {16987567} ORDER_TONE {} }

% $t marketByPriceSubmit add $DEPTHS

[Tclrfa::dispatchEventQueue] Event loop - approximate pending Events: 0

[Tclrfa::marketByPriceSubmit] mapAction: add

[Tclrfa::marketByPriceSubmit] symbol name: ANZ.CHA

[Tclrfa::marketByPriceSubmit] mapKey: 201000B

[Tclrfa::marketByPriceSubmit] fieldList:
ORDER_PRC=20.1000,ORDER_SIDE=BID,ORDER_SIZE=1300,NO_ORD=13,QUOTIM_MS=16987567,ORDER_TONE=,

[OMMCProvServer::submitData] sending refresh item: ANZ.CHA

[OMMCProvServer::submitData] sending refresh service: NIP
```

TS1

setTimeSeriesPeriod *daily|weekly|monthly*

Define a time period for a time series subscription. Default is daily.

setTimeSeriesMaxRecords *max_records*

max_records = number of maximum time series output records

Define the maximum record before calling `getTimeSeries`. Default is 10.

getTimeSeries *RIC*

RIC = Reuters Instrument Code

A helper function that subscribes, wait for data dissemination to be complete, unsubscribe from the service and return series as a list of records. `getTimeSeries` supports one time series retrieval at a time.

```
% set ric "CHK.N"

% set period "daily"

% set maxrecords "15"

% $t setTimeSeriesPeriod $period

% $t setTimeSeriesMaxRecords $maxrecords

% set timeseries [$t getTimeSeries $ric]

% puts "\n\n############### $ric $period ([llength $timeseries] records) ###############"

% puts [join $timeseries "\n"]\n\n


############### CHK.N daily (21 records) ###############
DATE,CLOSE,OPEN,HIGH,LOW,VOLUME,VWAP
2012/02/26,25.110,25.360,25.440,25.010,2457997.000,25.218
2012/02/23,25.450,24.830,25.830,24.820,3420367.000,25.451
2012/02/22,24.980,24.130,24.990,23.960,3786628.000,24.582
2012/02/21,24.030,24.220,24.340,23.640,4144242.000,23.998
2012/02/20,24.620,25.000,25.060,24.540,3713133.000,24.767
2012/02/19,Market holiday
2012/02/16,24.710,24.220,24.980,24.130,5116842.000,24.508
2012/02/15,23.770,23.070,23.790,22.730,3540557.000,23.489
2012/02/14,23.020,22.740,23.270,22.520,3885523.000,22.923
2012/02/13,22.710,22.800,22.950,22.450,3114849.000,22.684
2012/02/12,22.660,22.750,23.320,22.260,5041853.000,22.746
2012/02/09,22.130,21.900,22.190,21.560,3179328.000,21.970
2012/02/08,22.340,22.320,22.640,22.030,3298569.000,22.355
2012/02/07,22.110,22.310,22.510,22.030,2662700.000,22.187
2012/02/06,22.180,22.680,22.680,22.020,2829459.000,22.175
```

History

historyRequest *RIC ?RIC RIC ...?*

RIC = Reuters Instrument Code

Request for historical data (data domain 12), this domain is not officially supported by Thomson Reuters. User can define multiple items. The data dispatched through `dispatchEventQueue` in Tcl dict format as below:

Image

```
{<SERVICE_NAME> <ITEM_NAME> REFRESH } {<SERVICE_NAME> <ITEM_NAME> {<FID_NAME#1>
{<VALUE#1>} <FID_NAME#2> {<VALUE#2>} ... <FID_NAME#3> {<VALUE#X>}}}
```

Update

```
{<SERVICE_NAME> <ITEM_NAME> {<FID_NAME#1> {<VALUE#1>} <FID_NAME#2> {<VALUE#2>} ...
<FID_NAME#3> {<VALUE#X>}}}
```

```
% $t historyRequest "tANZ.AX"

% $t dispatchEventQueue

[HistoryHandler::processResponse] History Refresh: tANZ.AX.NIP

{NIP tANZ.AX REFRESH} {NIP tANZ.AX { TRDPRC_1 {40.124} SALTIM {18:34:30:216} TRADE_ID
{123456789} BID_ORD_ID {5307FBL20AL7B} ASK_ORD_ID {5307FBL20BN8A} }} {NIP tANZ.AX {
TRDPRC_1 {40.124} SALTIM {18:34:30:216} TRADE_ID {123456789} BID_ORD_ID {5307FBL20AL7B}
ASK_ORD_ID {5307FBL20BN8A} }} {NIP tANZ.AX { TRDPRC_1 {40.124} SALTIM {18:34:31:217}
TRADE_ID {123456789} BID_ORD_ID {5307FBL20AL7B} ASK_ORD_ID {5307FBL20BN8A} }}}
```

historyCloseRequest *RIC ?RIC RIC ...?*

RIC = Reuters Instrument Code

Unsubscribe items from historical data stream. The user can define multiple item names.

```
% $t historyCloseRequest “tANZ.AX”
```

historyCloseAllRequest

Unsubscribe all items from historical data stream.

```
% $t historyCloseAllRequest
```

historySubmit *data*

data = Tcl dict

For a provider client to publish the specified history data to MDH/ADH, each history data image/update must be in the following format:

`<ITEM_NAME> {<FID_NAME#1> {<VALUE#1>} ... {<FID_NAME#X>} <VALUE#X>}}`

```
% dict set UPDATES tANZ.AX { TRDPRC_1 {40.124} SALTIM {now} TRADE_ID {123456789}
BID_ORD_ID {5307FBL20AL7B} ASK_ORD_ID {5307FBL20BN8A} }

% $t historySubmit $UPDATES

[Tclrfa::historySubmit] symbolName: tANZ.AX

[Tclrfa::historySubmit] fieldList:
TRDPRC_1=40.124,BID_ORD_ID=5307FBL20AL7B,ASK_ORD_ID=5307FBL20BN8A,TRADE_ID=123456789,SA
TIM=now

[OMMCProvServer::submitData] sending update item: tANZ.AX

[OMMCProvServer::submitData] sending update service: NIP
```

Getting Data

dispatchEventQueue *?timeout?*

timeout = time in milliseconds to wait for data

Dispatch the events (data) from EventQueue within a period of time (If timeout is omitted, it will return immediately). If there are many events in the queue at any given time, a single call gets all the data until the queue is empty. Data is in a list of Tcl dicts.

```
% $t dispatchEventQueue

{NIP EUR= REFRESH} {NIP EUR= { RDNDISPLAY {100} RDN_EXCHID {SES} BID {0.988} ASK
{0.999} DIVPAYDATE {23 JUN 2011} }}
```