#!/dev/cartel

PyRFA 7.5

Reference Guide

5 November 2013 | www.devcartel.com

CONTENTS

ABOUT PYRFA

PyRFA is a python extension for accessing Thomson Reuters Enterprise Platform for Real-time (TREP-RT) or RMDS which is widely used in the financial market industry for delivery high-speed financial market data. It provides subscription and publication interfaces and capable of consuming/publishing various type of market data such as prices, order book, market depth, intraday history and time series data along with capabilities to consuming low latency or high throughput data rates.

FEATURES

- Subscription for MARKET_PRICE (level 1)
- Subscription for MARKET_BY_ORDER (order book)
- Subscription for MARKET_BY_PRICE (market depth)
- Snapshot request
- Dictionary download or use local files
- Directory request
- Symbol list request
- 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 PyRFA, please follow the steps below:

1. Download the latest PyRFA from http://devcartel.com/pyrfa
2. Extract the package on your server.
3. Open the `/example` folder
4. Configure `pyrfa.cfg` for `provider.py` and `consumer01.py` 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.py` with the command below, wait a few seconds for the data to be published to MDH/ADH cache:

```
$ python provider.py
```

6. Start `/example/consumer01.py` to consume the data from provider.py with the command below.

```
$ python consumer01.py
```

7. `consumer01.py` retrieves a market data full image followed by incremental updates of ‘EUR=’ and ‘C.N’ , below is the execution result:

```
[Thu Jul 04 17:23:57 2013]: (ComponentName) Pyrfa: (Severity) Information: [Pyrfa::login]
Login successful. (username: pyrfa)

(('NIP', 'C.N', 'REFRESH'), ('NIP', 'C.N', {'OPEN_TIME': '09:00:01:000', 'BID':

4.23, 'DIVPAYDATE': '23 JUN 2011', 'OFFCL_CODE': 'isin1234XYZ', 'RDN_EXCHID': 'NAS',
'RDNDISPLAY': 100}))

(('NIP', 'C.N', {'TIMACT': '17:24:00:354', 'ACVOL_1': 1031.0, 'TRDPRC_1': 4.149}),)

(('NIP', 'C.N', {'TIMACT': '17:24:00:854', 'ACVOL_1': 1032.0, 'TRDPRC_1': 4.555}),)
```

OMM DATA TYPE- PYTHON CONVERSION

OMM DATA TYPE	PYTHON
ENUM	STRING
FLOAT	DOUBLE
DOUBLE	DOUBLE
REAL32	DOUBLE
REAL64	DOUBLE
INT32	INTEGER
UINT32	INTEGER
INT64	LONG
UINT64	LONG

MODULE FUNCTIONS

Initialization

Pyrfa.**pyrfa()**

Instantiate a PyRFA object.

```
>>>p = Pyrfa.pyrfa()
```

Pyrfa.**setDebugMode([*Boolean*])**

Enable or disable debug messages. If argument is empty, it will read a value from configuration file.

```
>>>p.setDebugMode(True)
```

Configuration

Pyrfa.**createConfigDb(*String*)**

Read the configuration file where ***String*** can be full path or relative path. Initialize RFA context.

```
>>>p.createConfigDb("./pyrfa.cfg")

[Pyrfa::initializeRFA] Initializing RFA context and Python libraries...

[Pyrfa::createConfigDb] Loading RFA config from file: ./pyrfa.cfg
```

Pyrfa.**printConfigDb(*[String]*)**

Print a configuration node. If the input parameter ***String*** is omitted, this function returns all of the configuration values under the 'Default' namespace

```
>>>p.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

Pyrfa.**acquireSession(*String*)**

Acquire a session as defined in the configuration file where ***String*** is a session name. Look up for an appropriate connection and create a client-server network session. For example, Session1 is defined as follows:

```
\Sessions\Session1\connectionList = "Connection_RSSL1"
```

This module will create a session using "Connection_RSSL1".

```
>>>p.acquireSession("Session4")
```

Client

Pyrfa.**createOMMConsumer()**

Create an OMM consumer client.

```
>>>p.createOMMConsumer()
```

Pyrfa.**createOMMProvider()**

Create an OMM provider client.

```
>>>p.createOMMProvider()
```

Pyrfa.**login(*[username], [instanceId] , [applicationId]*)**

Send a login message through the acquired session. This step is mandatory in order to consume the market data from P2PS/ADS. If any argument is omitted, PyRFA will look it up from configuration file.

```
>>>p.login()

[Thu Jul 04 17:56:47 2013]: (ComponentName) Static: (Severity) Information: The

RSSL_Cons_Adapter initialization succeeded

...

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

Pyrfa.**isLoggedIn()**

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

```
>>>p.isLoggedIn()

True
```

Pyrfa.**setInteractionType(*type*)**

Set subscription ***type*** to be ‘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.

```
>>>p.setInteractionType(“snapshot”)
```

Directory

Pyrfa.**directoryRequest()**

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

```
>>>p.directoryRequest()
```

Pyrfa.**submitDirectory(*Long*)**

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 called automatically upon data submission

```
>>>p.submitDirectory(6)
```

Dictionary

Pyrfa.**dictionaryRequest()**

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

```
>>>p.dictionaryRequest()
```

Pyrfa.**isNetworkDictionaryAvailable()**

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

```
>>>p.isNetworkDictionaryAvailable()

True
```

Logging

Pyrfa.**log(*String*)**

Write a message to a log file.

```
>>>p.log('Print log message out')

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

Pyrfa.**logWarning(*String*)**

Write a warning message to a log file.

```
>>>p.logWarning('Print warning message out')

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

Pyrfa.**logError(*String*)**

Write an error message to a log file.

```
>>>p.logError('Print error message out')

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

Symbol List

Pyrfa.**symbolListRequest(*String*)**

For consumer application to subscribe symbol lists. User can define multiple symbol list names using "," to separate each name in ***String*** e.g. "ric1,ric2,ric3". Data dispatched through `dispatchEventQueue()` function in tuples:

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>','<COMMAND>',('ITEM_NAME', { '<FID_NAME#1>': <VALUE#1>, ...
,'<FID_NAME#X>':<VALUE#X>}))
```

```
>>> p.symbolListRequest("0#BMD")

>>> p.dispatchEventQueue()

(('NIP', '0#BMD', 'REFRESH'),('NIP', '0#BMD', 'ADD', ('FKLI', {}))),('NIP', '0#BMD',
'ADD', ('FCPO', {'PROD_PERM': 10, 'PROV_SYMB': 'MY439483'}))
```

Pyrfa.**symbolListCloseRequest(*String*)**

Unsubscribe the specified symbol lists. User can define multiple symbol list names using "," to separate each name in ***String***.

```
>>>p.symbolListCloseRequest("0#BMD")
```

Pyrfa.**symbolListCloseAllRequest()**

Unsubscribe all symbol lists.

```
>>>p.symbolListCloseAllRequest()
```

Pyrfa.**isSymbolListRefreshComplete()**

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

```
>>>p.isSymbolListRefreshComplete()

True
```

Pyrfa.**getSymbolList(*String*)**

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

```
>>>p.getSymbolList(“0#BMD”)

 FPCO FPKC FPRD FPGO
```

Pyrfa.**getSymbolListWatchList()**

Return names of the subscribed symbol Lists.

Example:

```
>>>p.getSymbolListWatchList()

 0#BMCA 0#ARCA
```

Pyrfa.**symbolListSubmit(*Command*, *Tuple*)**

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

- ADD - add new items to the symbol list
- UPDATE - update items in the symbol list
- DELETE - delete items from the symbol list

Tuple is an input tuple for which each symbol list must be in the following format:

```
('<SYMBOLLIST_NAME>', '<ITEM#1_NAME>', {'<FID_NAME#1>':<VALUE#1>,
'<FID_NAME#2>':<VALUE#2>, ... , ‘<FID_NAME#X>’:<VALUE#X>})
```

```
>>>SYMBOLLIST = (('0#BMD', 'FCPO', {'PROD_PERM':10, 'PROV_SYMB':'MY439483'}),)

>>>p.symbolListSubmit("ADD",SYMBOLLIST)

[Pyrfa::symbolListSubmit] mapAction: add

[Pyrfa::symbolListSubmit] symbolName: 0#BMD
```

```
[Pyrfa::symbolListSubmit] mapKey: FCPO

[Pyrfa::symbolListSubmit] fieldList: [Pyrfa::symbolListSubmit] fieldList:
PROV_SYMB=MY439483,PROD_PERM=10

[OMMCProvServer::submitData] sending refresh item: 0#BMD

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

Market Price

Pyrfa.**marketPriceRequest(*String*)**

For consumer client to subscribe market data from P2PS/ADS, user can define multiple item names using “,” to separate each name in ***String*** e.g “ric1,ric2”. The data dispatched through `dispatchEventQueue()` function in a tuple 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>}))
```

Example:

```
>>>p.marketPriceRequest(‘C.N’)

>>>p.dispatchEventQueue()

(('NIP', 'C.N', 'REFRESH'), ('NIP', 'C.N', {'OPEN_TIME': '09:00:01:000', 'BID': 4.23,
'DIVPAYDATE': '23 JUN 2011', 'OFFCL_CODE': 'isin1234XYZ', 'RDN_EXCHID': '123',
'RDNDISPLAY': 100}))

(('NIP', 'C.N', {'TIMACT': '18:52:38:551', 'ACVOL_1': 1262.0, 'TRDPRC_1': 4.28}),)
```

Pyrfa.**marketPriceCloseRequest(*String*)**

Unsubscribe items from streaming data. User can define multiple item names using “,” to separate each name.

```
>>>p.marketPriceCloseRequest (‘C.N,JPY=’)
```

Pyrfa.**marketPriceCloseAllRequest()**

Unsubscribe all items from streaming data.

```
>>>p.marketPriceCloseAllRequest()
```

Pyrfa.**getMarketPriceWatchList()**

Returns names of the subscribed items.

```
>>>p.getMarketPriceWatchList()

 C.N JPY=
```

Pyrfa.**marketPriceSubmit(*Tuple*)**

For provider client to publish market data to MDH/ADH, the market data image/update ***Tuple*** must be in the following Python tuple format:

```
('<ITEM_NAME>', {'<FID_NAME#1>':<VALUE#1>,...,'<FID_NAME#X>':<VALUE#X>})
```

```
>>>>IMAGE = ('EUR=', {'RDNDISPLAY':200, 'RDN_EXCHID':155, 'BID':0.988, 'ASK':0.999,
'DIVPAYDATE':'20110623'})

>>>p.marketPriceSubmit(IMAGE)

[Pyrfa::marketPriceSubmit] symbolName: C.N

[Pyrfa::marketPriceSubmit] fieldList: TRDPRC_1=4.201,ACVOL_1=1001

[OMMCProvServer::submitData] sending update item: C.N

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

Market by Order

Pyrfa.**marketByOrderRequest(*String*)**

For a consumer application to subscribe order book data, user can define multiple item names using "," to separate each name under ***String***. The data dispatched through `dispatchEventQueue`() module in a tuple below:

Images

```
((‘<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>}))
```

```
>>>p.marketByOrderRequest("ANZ.AX");

>>>p.dispatchEventQueue()

(('NIP', 'ANZ.AX', 'REFRESH')

('NIP', 'ANZ.AX', 'ADD', ('538993C200035057B', {'ORDER_SIDE': 'BID', 'ORDER_TONE':
'', 'SEQNUM_QT': 67.0, 'ORDER_PRC': 20.618000000000002, 'CHG_REAS': 10.0,
'ORDER_SIZE': 390.0, 'EX_ORD_TYP': 0.0})))
```

Pyrfa.**marketByOrderCloseRequest(*String*)**

Unsubscribe an item from order book data stream. User can define multiple item names using “,” to separate each name under ***String***.

```
>>>p.marketByOrderCloseRequest("ANZ.AX")
```

Pyrfa.**marketByOrderCloseAllRequest()**

Unsubscribe all items from order book data streaming service.

```
>>>p.marketByOrderCloseAllRequest()
```

Pyrfa.**getMarketByOrderWatchList()**

Return all subscribed item names on order book streaming data.

```
>>> p.getMarketByOrderWatchList()

 ANZ.AX
```

Pyrfa.**marketByOrderSubmit(*Command*, *Tuple*)**

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 order to the item
- DELETE - remove order from the item

Tuple is an input tuple of each order 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>})
```

```
ORDERS = (('ANZ.AX', '538993C200083483B', {'ORDER_PRC': '20.280', 'ORDER_SIZE':100,
'ORDER_SIDE':'ASK', 'SEQNUM_QT':2744, 'EX_ORD_TYP':0, 'CHG_REAS':6,'ORDER_TONE':''}),)

>>>p.marketByOrderSubmit('ADD', ORDERS)

[Pyrfa::marketByOrderSubmit] mapAction: update

[Pyrfa::marketByOrderSubmit] symbolName: ANZ.AX

[Pyrfa::marketByOrderSubmit] mapKey: 538993C200083483B

[Pyrfa::marketByOrderSubmit] fieldList: [Pyrfa::marketByOrderSubmit] fieldList:
EX_ORD_TYP=0,ORDER_SIZE=200,CHG_REAS=6,ORDER_PRC=20.982,SEQNUM_QT=2745,ORDER_TONE=,ORDE
R_SIDE=BID

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

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

Market by Price

Pyrfa.**marketByPriceRequest(*String*)**

For consumer application to subscribe market depth data, user can define multiple item names using "," to separate each name. 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>}))
```

```
>>>p.marketByOrderRequest("ANZ.CHA")

>>>p.dispatchEventQueue();

 (('NIP', 'ANZ.CHA', 'REFRESH'),

 ('NIP', 'ANZ.CHA', 'ADD', ('210000B', {'ORDER_SIDE': 'ASK', 'ORDER_TONE': '',
'ORDER_PRC': 21.0, 'NO_ORD': 13, 'QUOTIM_MS': 16987567, 'ORDER_SIZE': 500.0}))

 ('NIP', 'ANZ.CHA', 'UPDATE', ('201000B', {'ORDER_SIDE': 'BID', 'ORDER_SIZE': 41.0,
'NO_ORD': 6}))
```

Pyrfa.**marketByPriceCloseRequest(*String*)**

Unsubscribe an item from market depth data stream. User can define multiple item names using “,” to separate each name.

```
>>>p.marketByOrderCloseRequest("ANZ.AX")
```

Pyrfa.**marketByPriceCloseAllRequest()**

Unsubscribe all items from market depth streaming service.

```
>>>p.marketByPriceCloseAllRequest()
```

Pyrfa.**getMarketByPriceWatchList()**

Return all subscribed item names on market depth streaming data.

```
>>>p.getMarketByPriceWatchList()

 ANZ.CHA
```

Pyrfa.**marketByPriceSubmit(*Command*, *Tuple*)**

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 new order to the item
- UPDATE - update order to the item
- DELETE - remove order from the item

Tuple is an input tuple of each depth 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>})
```

```
>>>DEPTH = ('ANZ.CHA', '210000B', {'ORDER_PRC': price , 'ORDER_SIDE':'BID',
'ORDER_SIZE':size, 'NO_ORD':no_ord, 'QUOTIM_MS':16987567,'ORDER_TONE':''})

>>>p.marketByPriceSubmit('ADD',DEPTH)

[Pyrfa::marketByPriceSubmit] symbolName: ANZ.CHA

[Pyrfa::marketByPriceSubmit] mapKey: 210000B

[Pyrfa::marketByPriceSubmit] fieldList: [Pyrfa::marketByPriceSubmit] fieldList:
```

```
ORDER_SIZE=500,QUOTIM_MS=16987567,NO_ORD=13,ORDER_PRC=21.0000,ORDER_TONE=,ORDER_SIDE=ASK

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

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

TS1

setTimeSeriesPeriod(*String*)

Define a time period for a time series subscription. ***String*** can be one of “daily”, “weekly” or “monthly”. Default is “daily”.

setTimeSeriesMaxRecords(*Int*)

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

getTimeSeries(*String*)

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.

```
>>>ric = "CHK.N"

>>>period = "daily"

>>>maxrecords = 10

>>>p.setTimeSeriesPeriod(period)

>>>p.setTimeSeriesMaxRecords(maxrecords)

>>>timeseries = p.getTimeSeries(ric)

>>>print "\n\n############## " + ric + " " + period + " (" + str(len(timeseries)) +  " records) " + "##############"

>>>for record in timeseries:

    print record

############## CHK.N daily (11 records) ##############

DATE,CLOSE,OPEN,HIGH,LOW,VOLUME,VWAP
2013/11/03,28.840,27.980,29.050,27.950,1998632.000,28.666
2013/10/31,28.000,27.900,28.100,27.550,1027979.000,27.932
2013/10/30,27.960,28.190,28.270,27.680,1345424.000,28.005
2013/10/29,28.150,28.360,28.650,27.770,1370013.000,28.121
2013/10/28,28.320,28.260,28.500,28.210,1246324.000,28.325
2013/10/27,28.160,28.260,28.470,28.110,1328412.000,28.228
2013/10/24,28.470,28.410,28.680,28.153,2462643.000,28.447
2013/10/23,28.370,27.660,28.680,27.470,1773109.000,28.163
```

History

Pyrfa.**historyRequest(*String*)**

Request for historical data (RDM type 12), this domain is not officially supported by Thomson Reuters. User can define multiple item using “,” to separate each one under ***String***. The data dispatched through `dispatchEventQueue`() module in a tuple 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>}))
```

```
>>>p.historyRequest("tANZ.AX")

>>>p.dispatchEventQueue()

(('NIP',   'tANZ.AX',   'REFRESH'),('NIP',   'tANZ.AX',   {'SALTIM':   '23:02:07:255',
'TRADE_ID': '123456789', 'ASK_ORD_ID': '5307FBL20BN8A', 'BID_ORD_ID': '5307FBL20AL7B',
'TRDPRC_1': 40.124}))
```

Pyrfa.**historyCloseRequest(*String*)**

Unsubscribe items from historical data streaming service. The user can define multiple item names using “,” to separate each name under ***String***.

```
>>>p.historyCloseRequest("'tANZ.AX")
```

Pyrfa.**historyCloseAllRequest()**

Unsubscribe all items from historical data streaming service.

```
>>>p.historyCloseAllRequest()
```

Pyrfa.**historySubmit(*Tuple*)**

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

```
('<ITEM_NAME>', {'<FID_NAME#1>':<VALUE#1>,...,'<FID_NAME#X>':<VALUE#X>})
```

```
>>>UPDATES= ('tANZ.AX', {'TRDPRC_1':40.124, 'SALTIM':'now', 'TRADE_ID':'123456789',
'BID_ORD_ID':'5307FBL20AL7B', 'ASK_ORD_ID':'5307FBL20BN8A'})

>>>p.historySubmit(UPDATES)

[Pyrfa::historySubmit] symbolName: tANZ.AX

[Pyrfa::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

Pyrfa.**dispatchEventQueue(*[Timeout]*)**

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 tuple format.

```
>>>p.dispatchEventQueue()

(('NIP', 'C.N', 'REFRESH'), ('NIP', 'C.N', {'OPEN_TIME': '09:00:01:000', 'BID':4.23,
'DIVPAYDATE': '23 JUN 2011', 'OFFCL_CODE': 'isin1234XYZ', 'RDN_EXCHID': '123',
'RDNDISPLAY': 100}))
```