## **Tick Data Processor**

1. Main Features

* offers comprehensive [set of calculations](#FunctionByCategory) over Trade-and-Quotes (TAQ) timeseries data
* pre-processes NYSE TAQ files into binary indexed files to facilitate fast loading of timeseries TAQ data
* original data source can be replaced by alternative source of CTA/UTP Level 1 data
* consists of back-end server process, client API (python) and auxiliary utilities
* uses text-based message format for submitting service requests and receiving calculation results

2. Components

|  |  |
| --- | --- |
| Executable name | Description |
| taq-prep | Utility to convert NYSE TAQ psv files into binary indexed data files |
| taq-ctrl | Utility to examine content of data files |
| tick-calc | Tick processor server |

2.1 Preparing datafiles - taq-prep

Utility taq-prep can read input from stdin or from one or more files. Input data must be previously uncompressed.   
Example:

zcat EQY\_US\_ALL\_REF\_MASTER\_20200331.gz | taq-ctrl -d 20200331 -t master -s A  
zcat EQY\_US\_ALL\_TRADE\_20200331.gz | taq-ctrl -d 20200331 -t trade  
zcat SPLITS\_US\_ALL\_BBO\_A\_20200331.gz | taq-ctrl -d 20200331 -t quote -s A

Full list of program options:

|  |  |
| --- | --- |
| Option | Description |
| -d, --date ] | Specifies trade date; valid formats : YYYYMMDD or YYYY-MM-DD |
| -s, --symbol-group | Specifies symbol group; used for exclusively for quotes |
| -t, --file-type | Output file type. Valid values are : master, trade, quote or quote-po (price-only NBBO) |
| -i, --in-files | Comma-separated list of input files; if not present then data is read from stdin |
| -o, --out-dir | Output directory |

Produced datafiles are loaded on-demand by tick-calc server process in response to query request. Data file consists of 3 sections (one exception is master file , which has 2 sections)

* header, which defines record type, record size, number of symbols and total number of records
* data section contains actual data records, sorted by symbol and time
* symbol mapping section (not present in master file) contains locations of starting and ending records for all symbols

2.2. Viewing data file content - taq-ctrl

Utility taq-ctrl is used to view content of data file. By default, utility displays file header information and lists all symbols with corresponding record counts. Optionally comma-separated list of symbols can be specified, in which case entire record set for each symbol in the list will be dumped to stdout.

Example:  
Full list of program options:

$ ./taq-ctrl.exe -f 20200331.nbbo.X.dat --sort --pretty |head

date file 20200331.nbbo.X.dat  
file size 1,762,947,716  
record type Nbbo (with size)  
record size 32  
symbol count 93  
  
XLK 7,374,733  
XLU 4,798,575  
XLV 4,123,093  
XLE 3,797,234

$ ./taq-ctrl.exe -f 20200331.trd.dat --no-header -s "GYRO,LATN W"

GYRO,12:24:11.216971773,14.6,200,Q,Q,Y,Y,N  
GYRO,12:24:11.216977191,14.3,49,Q,Q,N,Y,N  
GYRO,12:24:11.216989442,14.6,200,Q,Q,N,N,N  
GYRO,16:00:00.046666054,14.6,200,P,P,N,N,N  
GYRO,16:00:00.514375721,14.3,49,Q,Q,N,N,N  
LATN W,14:51:59.172251037,0.25,100,Q,Q,Y,Y,N  
LATN W,14:51:59.172271648,0.25,100,Q,Q,N,N,N  
LATN W,16:00:00.089315406,0.25,100,P,P,N,N,N  
LATN W,16:00:00.536732544,0.25,100,Q,Q,N,N,N

|  |  |
| --- | --- |
| Option | Description |
| -f, --file | Path to datafile |
| -s, --symbols | Comma-separated symbol list |
| --no-header | Print output without header information |
| --sort | Sort symbols by record count in descending order |
| --pretty | Use pretty formatting |

2.3 Processing client requests - tick-calc

Main server process responsible for accepting client connections and executing query requests.  
Server locates and loads tick data files on-demand based on client request. After fulfilling requests the data is removed from application address space to keep memory utilization under control.

Full list of program options:

|  |  |
| --- | --- |
| Option | Description |
| -d, --data-dir | Path to datafiles |
| -l, --log-dir | Log file output directory |
| -t, --tcp | TCP port to accepts client connections |
| -c, --cpu | List of CPU cores e.g. 2,3,6-15 |
| --verbose | Additional messages written to stdout and log file |

3. API Refence

3.1 Python

Module name taqpy

* facilitates python code development
* handles network IO, hides details of communication protocol and internal data representation
* improves speed by implementing data formatting and parsing in C++
* includes helper routines for building valid service requests and processing query results

import taqpy

3.1.1 Query Execution

* connects to server and forwards query request followed by formatted input data
* blocks until response is received
* builds and returns response list

kwargs = {}  
for field in taqpy.ArgumentList(function\_name):  
 kwargs[field[0]] = np.array(req\_df[field[0]], dtype=field[1])

ret = taqpy.Execute(req\_json, \*\*kwargs)

ret\_df = None  
ret\_flds = taqpy.ResultFields(function\_name)  
if len(ret) > 1:  
 data = {}  
 for fld in range(len(ret\_flds)):  
 data[ret\_flds[fld][0]] = pd.Series(ret[fld+1])

ret\_df = pd.DataFrame(data)

|  |  |
| --- | --- |
| Function Name | Execute |
| Expected Arguments | First argument is always request in [JSON format](#RequestFormat). followed by keyworded, variable-length argument list, each represented by a numpy array with expected data type and size.  Full list of argument names and datatypes can be retrieved using [auxiliary functions](#AuxFunctions) listed in next section. |
| Return Values | List with one or more elements.  First element is always execution summary in [JSON format](#ExecutionSummaryFormat). In case of successful execution list will have additional entries , each represented by a numpy array that corresponds to a calculated field. Full list of field names and datatypes can be retrieved using [auxiliary functions](#AuxFunctions) listed in next section. |

3.1.2 Auxiliary routines

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Arguments | Return Value | Description |
| Describe |  | Text in JSON format: *default\_tz* default time zone *input\_fields*  list of names and datatypes  *output\_fields* list of names and datatypes | Describes all supported functions, corresponding arguments and result datasets |
| FunctionList |  | List of function names | Lists all supported functions |
| ArgumentList | function name | List of fields names | Lists of arguments by name |
| ArgumentNames | function name | List of fields names and datatypes | Lists arguments’ names and datatypes |
| ResultFields | function name | List of fields names and datatypes | Lists output fields’ names and datatypes |

>>> taqpy.FunctionList()  
['Quote', 'ROD']

>>> taqpy.ArgumentNames('Quote')  
['Symbol', 'Timestamp']

>>> taqpy.ArgumentList('Quote')  
[('Symbol', 'a18'), ('Timestamp', 'a36')]

>>> taqpy.ResultFields('Quote')  
[('ID', 'int'), ('Timestamp', 'a36'), ('BestBidPx', 'float'), ('BestBidQty', 'a4'), ('BestOfferPx', 'float'), ('BestOfferQty', 'a4')]

>>>

3.2 Alternative API

Text-based message format is used for communication between client and server applications. Therefore, a client application can be relatively easy coded in any language as long as it offers basic network I/O services and simple text parsing and data conversion routines.

Example of message exchange between client and server applications

$ socat -v TCP-LISTEN:3090,fork,reuseaddr TCP:127.0.0.1:3091

> 2020/07/29 23:02:52.006072 length=430 from=0 to=429  
{ "tcp": "127.0.0.1:3090", "request\_id": "273edddc-d211-11ea-90c8-6c96cff12815", "function\_list": [ "Quote" ], "argument\_list": [ "Symbol",

"Timestamp" ], "separator": "|", "input\_sorted": "true", "input\_cnt": "3", "output\_format": "psv", "time\_zone": "America/New\_York"}  
BAC|2020-03-31T10:54:36.745180000  
SPY|2020-03-31T14:56:41.377343000  
AMZN|2020-03-31T11:10:02.335551000  
< 2020/07/29 23:02:53.165026 length=519 from=0 to=518  
{ "request\_id": "273edddc-d211-11ea-90c8-6c96cff12815", "output\_fields": [ "ID", "Timestamp", "BestBidPx", "BestBidQty", "BestOfferPx",

"BestOfferQty" ], "output\_records": "3", "error\_summary": "", "runtime\_summary": { "parsing\_input": "00:00:00", "execution": "00:00:01.0006380

00", "sorting\_output": "00:00:00" }}  
1|10:54:36.691733326|22.28|184|22.29|32  
2|14:56:41.354295735|260.11|6|260.12|37  
3|11:10:02.295226078|1974.59|2|1975.9|3

Appendix A. Request Format

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| request\_id | String | Unique ID |
| tcp | String | IPv4 address in doted notation and TCP port separated by colon |
| function\_list | Array | List of functions |
| argument\_list | Array | List of arguments |
| separator | String | Character, a separator used in input records’ formatting |
| time\_zone | String | ISO name of time zone; supported values "UTC" and "America/New\_York" |
| input\_sorted | Boolean | Indicates if records within symbol/date group are sorted by timestamp True signals to server that sorting of input records is not necessary |
| input\_cnt | Number | Integer, number of input records |
| output\_format | String | Currently "psv" or "csv" |

Example

{  
 "request\_id": "07ff4f8a-d056-11ea-a52b-6c96cff12815",  
 "tcp": "127.0.0.1:3090",  
 "function\_list": [  
 "Quote"  
 ],  
 "argument\_list": [  
 "Symbol",  
 "Timestamp"  
 ],  
 "separator": "|",  
 "time\_zone": "America/New\_York",  
 "input\_sorted": True,  
 "input\_cnt": 100,  
 "output\_format": "psv"  
}

Appendix B. Execution Summary Format

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| request\_id | String | , unique ID taken from original request |
| output\_fields | Array | , List of output fields |
| output\_records | Number | Number of records in result set |
| error\_summary | Array | List of errors if present,  Each entry is represented by a tuple (JSON Object) : error type and count |
| runtime\_summary | Object | shows time spent by server for parsing/sorting input records, execution requested calculation function, and sorting/merging output records |

Example

{  
 "request\_id": "07ff4f8a-d056-11ea-a52b-6c96cff12815",  
 "output\_fields": [  
 "ID",  
 "Timestamp",  
 "BestBidPx",  
 "BestBidQty",  
 "BestOfferPx",  
 "BestOfferQty"  
 ],  
 "output\_records": "67",  
 "error\_summary": [{  
 "type": "DataNotFound",  
 "count": "33"  
 }],  
 "runtime\_summary": {  
 "parsing\_input": "00:00:00.000988000",  
 "execution": "00:00:01.000973000",  
 "sorting\_output": "00:00:00"  
 }  
}

Appendix C. Functions by Category

|  |  |  |
| --- | --- | --- |
| Category Name | Function Name | Description |
| Trades | Last Trade | Returns latest eligible trade for given time |
| Trades | Volume | Returns total volume, unconstrained VWAP, trade count, min/max/avg/stddev price and min, max size calculated between given start and time Optional print type constrain can be specified; default is d)   1. all eligible trades 2. exclusively TRF normal prints 3. exclusively exchange-only normal prints or 4. both i.e. all tape prints as results CLOB executions |
| Trades | Volume By Tick | Same as above, except start or end time is returned instead if trade count  Period end is determined by tick count  Negative tick count changes direction of calculation backward in time |
| Trades | VWAP | Calculates VWAP between start and end times Accepts 2 optional constraints: - price (side and limit price); default – none  - print type constrain |
| Trades | VWAP By Volume | Same as above  Quantity and target participation rate are used to determine period end time |
| Trades | VWAP By Tick | Same as above  Period end is determined by tick count  Negative tick count changes direction of calculation backward in time |
| Trades | Best Price | Calculates best possible VWAP by selecting most favorable trades between start and end times, enough to fill required quantity |
| Trades | Best Price POV | Same as above Required quantity and target participation rate are used to determine period end time |
| Trades | RPM | Returns relative performance measure given order average execution price and start and end times |
| Quotes | Quote | Returns latest valid quote for given time |
| Quotes | Time Weighted Quote | Returns time-weighted bid, offer and spread for given time period |
| Quotes | ROD | Resting order duration (multirow input) |
| Reference | Security Master Record | Returns security master record(s) |

Appendix D. C++ Build Dependencies

List of commands for installing and building necessary s/w on CentOS 7 hosts

# 1 install utilities  
yum install git  
yum install cmake3  
yum install wget  
cd /usr/bin/  
sudo ln -s cmake3 cmake  
yum install python3-devel

# 2 install compiler  
yum install centos-release-scl  
yum install devtoolset-8-gcc devtoolset-8-gcc-c++  
add "source scl\_source enable devtoolset-8" to .bashrc  
pip3 install pytest #used by many public github distributions

# 3 boost  
mkdir /opt/Toolbox  
cd /opt/Toolbox  
wget <https://dl.bintray.com/boostorg/release/1.72.0/source/boost_1_72_0.tar.gz>  
tar -zxvf boost\_1\_72\_0.tar.gz  
cd boost\_1\_72\_0  
./bootstrap.sh # to configure build system for current platform  
./b2 # to build C++ boost libraries

# 4 pybind11  
cd /opt/Toolbox  
git clone <https://github.com/pybind/pybind11.git>  
cd pybind11  
mkdir build  
cd build  
cmake -DBoost\_INCLUDE\_DIR=/opt/Toolbox/boost\_1\_72\_0 ..  
make -j8