Tick Data Processor User Guide

|  |  |  |
| --- | --- | --- |
| Date | Document version | Changes |
| 01/05/2020 | 1.0 | Initial draft |
|  |  |  |

Table of Contents

[Introduction 3](#_Toc52865640)

[Application Architecture 3](#_Toc52865641)

[Input Data 3](#_Toc52865642)

[Data File Format 4](#_Toc52865643)

[Data File Management 4](#_Toc52865644)

[Main Server Process 5](#_Toc52865645)

[Wire Protocol 6](#_Toc52865646)

[Request Format 6](#_Toc52865647)

[Execution Summary Format 6](#_Toc52865648)

[Data records 7](#_Toc52865649)

[Client Application 8](#_Toc52865650)

[Design considerations 8](#_Toc52865651)

[Python API 8](#_Toc52865652)

[Astral API 8](#_Toc52865653)

[Use Cases 8](#_Toc52865654)

[Factsheet 8](#_Toc52865655)

[Performance Library 8](#_Toc52865656)

[Astral API Standard Data 8](#_Toc52865657)

[Astral API extension to Standard Data 9](#_Toc52865658)

[Astral API extension for TAQ 9](#_Toc52865659)

[Python API 9](#_Toc52865660)

[Query Execution 9](#_Toc52865661)

[Auxiliary routines 10](#_Toc52865662)

[TDP Reference 11](#_Toc52865663)

[Quotes 11](#_Toc52865664)

[NBBO 11](#_Toc52865665)

[NBBOPrice 11](#_Toc52865666)

[Time Weighted NBBO 12](#_Toc52865667)

[Resting Order Duration (ROD) 12](#_Toc52865668)

[Trades 13](#_Toc52865669)

[VWAP 13](#_Toc52865670)

[Best Price 14](#_Toc52865671)

[Last Trade 15](#_Toc52865672)

[Relative Performance Measure (RPM) 15](#_Toc52865673)

[Reference 15](#_Toc52865674)

[Security Master Record 15](#_Toc52865675)

[Hybrid 16](#_Toc52865676)

[Appendices 16](#_Toc52865677)

[Appendix A. Files Formats 16](#_Toc52865678)

[Binary File Header 16](#_Toc52865679)

[Security Master Record 16](#_Toc52865680)

[Trade Record 17](#_Toc52865681)

[NBBO Record 17](#_Toc52865682)

[NBBO Price-only Record 17](#_Toc52865683)

[Appendix B. Data Formats 18](#_Toc52865684)

[Appendix C. VWAP Flavors 18](#_Toc52865685)

[Appendix D. Markouts 18](#_Toc52865686)

[Appendix E. C++ Build Dependencies 19](#_Toc52865687)

[Appendix F. Build Process 20](#_Toc52865688)

[Appendix G. Installation Process 20](#_Toc52865689)

# Introduction

Tick Data Processor (TDP) is application designed to perform set of calculations over time-series of historic trading data.

The main purpose of this application is to replace or simplify process presently used for producing Astral Factsheet reports, and offer improved platform for development of new products, which in some form rely on historic tick data.

## Application Architecture

The primary function of TDP is providing users with a tool for annotating transactional data with market data-based calculations utilizing library of functions commonly used in execution analysis.

Product design was influenced by following factors:

* Processing of NYSE TAQ files in original workflow for Astral Factsheet is expensive. Demand for memory, storage, and CPU resources is high, and greatly contributes to costs.
* The process is slow and unstable. Attempts to improve performance resulted in overengineered software, which adds less obvious, yet considerable costs in form of product maintenance.
* New application must address main design flaw of original implementation which causes movement of large quantities of raw tick data across cloud's computing nodes.
* Source of historic market data is not finalized. The selection will be made based on several consideration, such as quality of data, price, and dissimilation rights. In future the selected source might be replaced with superior alternative.

TDP application consists of back-end server process, client API for Python, and auxiliary utilities.

Table 1. Components of Tick Data Processor

|  |  |
| --- | --- |
| 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 |
| taqpy | Python API extension |
| taqproc\_testkit.py | Testing toolkit; facilitates development of unit tests |
| sample application | Python sample scripts; illustrate usage of Python API |

# Input Data

The first step in new workflow is converting NYSE TAQ text datafiles into binary files. The job of converting files is performed during night cycle when NYSE TAQ files are downloaded from NYSE sftp site, and subsequently pre-processed with taq-prep utility

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

Full list of taq-prep program options:

|  |  |
| --- | --- |
| Option | Description |
| -d, --date | Specifies trade date; valid formats: YYYYMMDD or YYYY-MM-DD |
| -s, --symbol-group | Specifies symbol group; used 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 |

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

There are 4 types of binary files:

* Security Master file, one file for each trading day, contains one record per ticker symbol (see Appendix)
* Trades, one file a day, contains all trades reported to CTA
* NBBO, one file per symbol group (26 a day), contains NBBO changes price and size; symbol group is defined by first letter of ticker symbol
* NBBO Price Only, one file per symbol group, contains only NBBO price changes ignoring size

After completion of input data conversion step the resulting binary files are ready for use and can loaded on-demand by tick-calc server process in response to query requests.

## Data File Format

Produced data files consists of 3 sections (one exception is security master file, which has 2 sections)

* the 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

# Data File Management

Data files are stored on disk in directory tree, where each trading date has dedicated sub-directory. Sub-directories are named using YYYYMMDD format.

Contents of data files can be examined with help of taq-ctrl utility. 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 written to stdout.

Full list of taq-ctrl program options:

|  |  |
| --- | --- |
| Option | Description |
| -f, --file | Path to datafile |
| -s, --symbols | Comma-separated symbol list |
| --no-header | Suppresses output of header information |
| --sort | Sort symbols by record count in descending order |
| --pretty | Use pretty formatting |

Example:

$ taq-ctrl -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

# Main Server Process

Executable file for main server process is tick-calc. The server is responsible for accepting client connections and executing query requests. Required tick data files are loaded by the server. After fulfilling requests, unreferenced data is removed from application address space to keep memory utilization under control.

TDP achieves its performance goals by processing requests in memory having all related market data records located in adjacent addresses as time series. Input record set is divided into subtasks with unique symbol/date combination, which are executed in parallel. Each subtask is pinned to dedicated CPU core allowing optimal use of system's data caches.

Full list of tick-calc 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 |

Description of all supported functions can be viewed in TDP Reference

## Wire Protocol

Tick Data Processor uses text-based data representation for sending and receiving data.  
Query request consists of one of more lines of JSON-formatted request header, followed by one or more input records separated by new line character.  
Result record set consist of one line of JSON-formatted response header followed by zero or more result records , also separated by new line character.

## Request Format

|  |  |  |
| --- | --- | --- |
| Field Name | JSON 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": [  
 "NBBO"  
 ],  
 "argument\_list": [  
 "Symbol",  
 "Timestamp"  
 ],  
 "separator": "|",  
 "time\_zone": "America/New\_York",  
 "input\_sorted": True,  
 "input\_cnt": 100,  
 "output\_format": "psv"  
}

## Execution Summary Format

|  |  |  |
| --- | --- | --- |
| Field Name | JSON 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"  
 }  
}

## Data records

Data records in both record sets, for request and for result, are separated by new-line and represented as tuples with single-character separator, '|' by default. Fields in tuples defined in JSON: "argument\_list" in case of request, and "output\_fields" in response.

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": [ "NBBO" ], "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

# Client Application

## Design considerations

Client applications can successfully interact with TDP if programming environment offers basic network I/O services, and simple text parsing and data conversion routines. Therefore, it can be written almost in any programming language.

Handling all communications directly is certainly the least restrictive option from programmer's point of view, however it might be unavailable for some reasons e.g. it is not available over Internet.

I addition to direct communication with the server TDP offers two more methods: Python API and Astral API.

## Python API

Python API is implemented as taqpy.so shared library extension. This method achieves performance benefits by using C++ to format service requests, allocate memory and parse inbound result records. It hides communication protocol and data representation.

## Astral API

This method exposes TDP query service to multitude of client application by offering REST API wrapper around taqpy.  
It offers a single end point for user authentication and access to all data sources within Astral cloud.

## Use Cases

### Factsheet

Current use by Enricher is limited to two functions:

1. Retrieval of quote to determine midpoint as reference price at order's arrival time
2. Calculate resting order duration measure according to algorithm that uses NBBO data

Because Enricher is inaccessible outside of Astral cloud its implementation does not need access control. All what necessary is a connectivity i.e. open TCP/IP address and port.

Suggested method: taqpy Python extension

### Performance Library

Performance library offers set of advanced calculations based on transactional data input. Transactional data is sourced via Astral API; therefore, it makes sense to use the same method to retrieve related calculations from TDP.

Suggested method: Astral API

### Astral API Standard Data

Standard data schema can be extended to include commonly used tick data calculations, such as reference prices, to make product more useful to clients. Because calculations are standard and limited in numbers, it makes sense to store values together with standard transactional data. Presence of additional fields in database enables clients to use them in filters and aggregations.

Suggested method: taqpy Python extension while processing transactional data and store results together in the same database.

### Astral API extension to Standard Data

Astral API can be extended to offer client ability to create custom result fields by calling TDP functions and passing elements of transactional data as input arguments.

There is opportunity to hide some complexity by adding fields that are well understood and identified, in which case required arguments can be inferred from other fields in the schema e.g. "Arrival Quote" is relevant to orders only and is the latest valid quote at the time of order arrival , therefore user might be able to simply select this field , and API implementation would pass correct arguments from the order record.

### Astral API extension for TAQ

Astral API can be modified to expose TDP functionality where input records are passed via REST API rather than from results of the Standard Data query.

# Python API

Module name taqpy

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

import taqpy

## Query Execution

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

|  |  |
| --- | --- |
| Function Name | Execute |
| Expected Arguments | First argument is always request in JSON format.  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 listed in next section. |
| Return Values | List with one or more elements.  First element is always execution summary in JSON format.  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 listed in next section. |

Example

kwargs = {}

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

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

ret\_json = json.loads(ret[0])

ret\_flds = ret\_json['output\_fields']

ret\_df = None

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)

## 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 tuples: field name, datatypes and required flag | Lists arguments’ names and datatypes |
| ArgumentNames | function name | List of fields names and datatypes | Lists of arguments by name |
| ResultFields | function name | List of fields names and datatypes | Lists output fields’ names and datatypes |

Example

>>> taqpy.FunctionList()  
['NBBO', 'NBBOPrice', 'ROD', 'VWAP']

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

>>> taqpy.ArgumentList('NBBO')  
[('Symbol', 'a18', True), ('Timestamp', 'a36', True), ('Markouts', 'a96', False)]

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

>>> taqpy.ResultFields('NBBOPrice')  
[('ID', 'int'), ('Timestamp', 'a36'), ('BestBidPx', 'float'), ('BestOfferPx', 'float')]

>>>

# TDP Reference

Functions by Category

## Quotes

### NBBO

Returns latest valid NBBO record for given symbol and timestamp.  
Valid record is a record with both sides of NBBO quote present.

Arguments

|  |  |  |
| --- | --- | --- |
| Argument Name | Data Type | Comments |
| Symbol | [symbol](#DataFormat_Symbol) |  |
| Timestamp | [timestamp](#DataFormat_Timestamp) | if TAQ data does not contain valid NBBO record with exact time, then previous valid record is returned |
| Markouts | a96 | optional; comma-separated string with one or more markout durations relative to reference time ([Appendix D](#_Appendix_D._Markouts)) |

Result fields

|  |  |  |
| --- | --- | --- |
| Filed Name | Data Type | Comments |
| ID | integer | input record number in original order, starting from 1 |
| Timestamp | timestamp | timestamp of NBBO record |
| BestBidPx | double |  |
| BestBidQty | integer |  |
| BestOfferPx | double |  |
| BestOfferQty | integer |  |

Presence of Markouts argument creates N sets of additional result fields, one for each element in markouts array.  
E.g. with value '1t,5t,10t' the result contains 1+5+4x5 fields:  
ID - record ID of the request as its index in the input record set starting with 1  
BestBidPx BestBidQty BestOfferPx BestOfferQty - correspond to NBBO at Timestamp  
BestBidPx\_1 BestBidQty\_1 BestOfferPx\_1 BestOfferQty\_1 - correspond to NBBO at Timestamp + 1 tick  
. . .   
BestBidPx\_4 BestBidQty\_4 BestOfferPx\_4 BestOfferQty\_4 - correspond to NBBO at Timestamp + 10 ticks

### NBBOPrice

Similar function to NBBO.  
Function arguments are the same, however result fields are not. Result fields include only timestamp and prices.

Result fields

|  |  |  |
| --- | --- | --- |
| Filed Name | Data Type | Comments |
| ID | integer | input record number in original order, starting from 1 |
| Timestamp | timestamp | timestamp of NBBO record |
| BestBidPx | double |  |
| BestOfferPx | double |  |

### Time Weighted NBBO

Measures are calculated over specified period. Each quote is weighted by the time it was in force. Result is average values for bid, offer and spread.

Arguments

|  |  |  |
| --- | --- | --- |
| Argument Name | Data Type | Comments |
| Symbol | symbol |  |
| Date | [date](#DataFormat_Date) |  |
| StartTime | time |  |
| EndTime | time |  |
| Markouts | a96 | optional |

Result fields

|  |  |  |
| --- | --- | --- |
| Filed Name | Data Type | Comments |
| ID | integer | input record number in original order, starting from 1 |
| TwaBid | double | time-weighted average bid |
| TwaOffer | double | time-weighted average offer |
| TwaSpread | double | time-weighted average spread |

### Resting Order Duration (ROD)

Function returns resting duration calculations for set of orders. Resting order duration measure is expressed by number of shares (LeavesQty) multiplied by duration in seconds.  
Input data for an order can be represented by one or more records. A record may or may not have execution-related fields populated.  
Typical scenarios are:

1. Order was not executed.  
   Expect one record with start and end times populated, with no execution time.
2. Order was filled on arrival.  
   Expect one record with start and end times populated, plus execution time and size, which should be equal to order size.  
   In this scenario execution time overrides end time.
3. Order had multiple executions.  
   Expect more than one record, likely equal to execution count, or execution count + 1.  
   End time correction applies in case when order is fully executed using time of the last fill.

During calculation entire order duration from start to end time is broken into segments, defined by 2 types of events: NBBO changes and order executions. Result of calculation is represented by 7 quantities, each corresponding to a resting band. Resting bands are defined by position of order’s working price relative to both sides of current NBBO.   
Bands are identified by integer values in -3:3 range, where -3 is the least aggressive e.g. for buy order value of -3 corresponds to working price below bid, and 3 is higher than offer.

Additional details can be found in Factsheet product specification.

Arguments

|  |  |  |
| --- | --- | --- |
| Argument Name | Data Type | Comments |
| ID | a64 | unique order identifier most reliable ID is combination of MPID and OrderID |
| Symbol | symbol |  |
| Date | date |  |
| StartTime | time |  |
| EndTime | time |  |
| Side | side |  |
| OrdQty | integer |  |
| LimitPx | double |  |
| MPA | double | maximum working price aggressiveness as band id |
| ExecTime | time |  |
| ExecQty | integer |  |

Result fields

|  |  |  |
| --- | --- | --- |
| Filed Name | Data Type | Comments |
| ID | a64 | unique order identifier |
| MinusThree | double | shares\* seconds when working price (WP) is less aggressive than near-touch |
| MinusTwo | double | shares\* seconds WP at near |
| MinusOne | double | shares\* seconds WP between near and mid-point |
| Zero | double | shares\* seconds WP at mid-point |
| PlusOne | double | shares\* seconds WP between mid-point and far-touch |
| PlusTwo | double | shares\* seconds WP at far |

## Trades

### VWAP

Returns VWAP for set of orders calculated over duration from start time until end time.  
There are 3 ways to define end time:

1. explicit by providing end time
2. based on target volume and participation - time required to fill target volume while participating at target rate
3. trade (tick) count

TAQ execution record is eligible for inclusion in VWAP calculation only if it contributes to daily volume. Using Flavor argument user can further refine eligibility of executions.

Arguments

|  |  |  |
| --- | --- | --- |
| Argument Name | Data Type | Comments |
| Symbol | symbol |  |
| Date | date |  |
| StartTime | time | first trade is the latest record with timestamp less than or equal to start time |
| Side | [side](#DataFormat_Side) | optional, if missing VWAP is calculated without price constraint |
| LimitPx | double | optional, if missing VWAP is calculated without price constraint |
| Flavor | integer | optional; specifies which eligible trades are included into VWAP calculation  1 - only normal trades reported by exchanges 2 - only normal trades reported by alternative display facility (off-exchange volume) 3 - normal trades (default)  4 - all eligible trades  5 - block trades (10+K size or $200K notional) |
| EndTime | time | optional, explicit end time |
| TargetVolume | integer | optional, positive integer |
| TargetPOV | double | optional, participation as value in 0-1 range |
| Ticks | integer | optional, negative value will use TAQ records that ends with start time |
| Markouts | a96 | optional |

Result fields

|  |  |  |
| --- | --- | --- |
| Filed Name | Data Type | Comments |
| ID | integer | input record number in original order, starting from 1 |
| TradeCnt | integer | number of executions included into VWAP calculation |
| TradeVolume | integer | volume included into VWAP calculation |
| VWAP | double | VWAP |

Presence of Markouts argument creates N sets of result fields, one for each element in markouts array.  
E.g. with '1t,5t,10t' the names of fields in result set are: ID, TradeCnt\_1, VWAP\_1, TradeVolume\_1, TradeCnt\_2 .. TradeVolume\_4, VWAP\_4

### Best Price

Calculates best possible VWAP by selecting most favorable trades between start and end times. Required volume should be enough to fill required quantity

Arguments

|  |  |  |
| --- | --- | --- |
| Argument Name | Data Type | Comments |
| Symbol | symbol |  |
| Date | date |  |
| StartTime | time |  |
| EndTime | time | optional |
| TargetVolume | integer | required volume |
| TargetPOV | double | optional, participation as value in 0-1 range |
| Side | side |  |
| LimitPx | double | optional, if missing VWAP is calculated without price constraint |

Result fields

|  |  |  |
| --- | --- | --- |
| Filed Name | Data Type | Comments |
| ID | integer | input record number in original order, starting from 1 |
| TradeCnt | integer | number of executions included into VWAP calculation |
| TradeVolume | integer | volume included into VWAP calculation |
| VWAP | double | VWAP |

### Last Trade

Returns last eligible trade on or before given timestamp.

Arguments

|  |  |  |
| --- | --- | --- |
| Argument Name | Data Type | Comments |
| Symbol | symbol |  |
| Timestamp | timestamp |  |
| Flavor | integer | optional |

Result fields

|  |  |  |
| --- | --- | --- |
| Filed Name | Data Type | Comments |
| ID | integer | input record number in original order, starting from 1 |
| Timestamp | timestamp | time of execution |
| ExecPx | double |  |
| ExecQty | integer |  |

### Relative Performance Measure (RPM)

Relative performance of executed order is calculated using trade data reported between order's start and end times. All executions prices are compared with order's average fill price. RPM is average of two measures: percentage of trades executed at a price not better than fill price, and percentage of corresponding volume.

Arguments

|  |  |  |
| --- | --- | --- |
| Argument Name | Data Type | Comments |
| Symbol | symbol |  |
| Date | date |  |
| StartTime | time |  |
| EndTime | time |  |
| Side | side |  |
| ExecPx | double | order's average fill price |

Result fields

|  |  |  |
| --- | --- | --- |
| Filed Name | Data Type | Comments |
| ID | integer | input record number in original order, starting from 1 |
| RPM | double | value in 0-1 range |

## Reference

### Security Master Record

Returns security master records

Arguments

|  |  |  |
| --- | --- | --- |
| Argument Name | Data Type | Comments |
| Symbol | symbol |  |
| Date | date |  |

Result fields: TBD

## Hybrid

TBD

# Appendices

## Appendix A. Files Formats

### Binary File Header

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Offset | Size in bytes | Description |
| size | integer | 0 | 4 | size of the header record |
| type | enum | 4 | 4 | type of data records e.g. NBBO |
| version | integer | 8 | 4 | version number (reserved for future use) |
| date | Date | 12 | 4 | trading date |
| symb\_cnt | integer | 16 | 4 | number of symbols |
| rec\_cnt | integer | 20 | 4 | number of data records |

### Security Master Record

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Offset | Size in bytes | Description |
| symb | character | 0 | 18 | copied form [NYSE TAQ](https://www.nyse.com/publicdocs/nyse/data/Daily_TAQ_Client_Spec_v3.0.pdf) file |
| utp\_symb | character | 18 | 18 | UTP symbol in accordance with [Nasdaq ticker symbol convention](https://www.nasdaqtrader.com/trader.aspx?id=CQSsymbolconvention) |
| CUSIP | character | 36 | 10 | copied form NYSE TAQ file |
| type | character | 46 | 4 | copied form NYSE TAQ file |
| sip\_symb | character | 50 | 18 | copied form NYSE TAQ file |
| prev\_symb | character | 68 | 18 | copied form NYSE TAQ file |
| test\_flag | character | 86 | 1 | copied form NYSE TAQ file |
| exch | character | 87 | 1 | copied form NYSE TAQ file |
| tape | character | 88 | 1 | copied form NYSE TAQ file |
| trd\_unit | integer | 89 | 1 | copied form NYSE TAQ file |
| lot\_size | integer | 90 | 1 | copied form NYSE TAQ file |
| industry\_code | character | 91 | 5 | copied form NYSE TAQ file |
| halt\_reason | character | 96 | 1 | copied form NYSE TAQ file |
| shares\_outstanding\_m | double | 104 | 8 | copied form NYSE TAQ file |
| exch\_mask | bit set | 112 | 8 | copied form NYSE TAQ file each bit corresponds to exchange CTA exchange code e.g. 0 bit - 'A' - NYSE American etc. Full list of CTA exchange code can be found in [NYSE Daily TAQ file specification](https://www.nyse.com/publicdocs/nyse/data/Daily_TAQ_Client_Spec_v3.3.pdf) |
| volume\_total | integer | 120 | 8 | calculated, total daily volume |
| volume\_regular | integer | 128 | 8 | calculated, daily volume limited to regular trades |
| volume\_trf | integer | 136 | 8 | calculated, daily off-exchange volume (TRF prints) |
| volume\_block | integer | 144 | 8 | calculated, daily volume limited to block trades |
| volume\_pre\_open | integer | 152 | 8 | early morning session's volume |
| volume\_post\_close | integer | 160 | 8 | late night session's volume |
| open\_time\_primary | Time | 168 | 8 | primary exchange main session open auction time |
| open\_price\_primary | double | 176 | 8 | primary exchange main session open auction price |
| open\_volume\_primary | integer | 184 | 8 | primary exchange main session open auction volume |
| close\_time\_primary | Time | 192 | 8 | primary exchange main session closing auction time |
| close\_price\_primary | double | 200 | 8 | primary exchange main session closing auction price |
| close\_volume\_primary | integer | 208 | 8 | primary exchange main session closing auction volume |
| average\_trade\_size | integer | 216 | 4 | calculated, average size of regular trade |
| min\_price\_regular | double | 224 | 8 | calculated, lowest price of regular trade |
| max\_price\_regular | double | 232 | 8 | calculated, highest price of regular trade |
| vwap\_regular | double | 240 | 8 | calculated, daily average volume-weighted price of regular trades |

### Trade Record

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Offset | Size in bytes | Description |
| time | Time | 0 | 8 | execution time |
| price | double | 8 | 8 | execution price |
| qty | integer | 16 | 4 | execution quantity |
| cond | char | 20 | 4 | CTA sale condition |
| exch | byte | 24 | 1 | CTA exchange code |
| trf | byte | 1 | CTA TRF code |
| lte | bit | 1 | flag indicating that the trade reset last price |
| ve | bit | flag indicating that trade contributes to daily volume |
| primary\_session | bit | flag indicating that trade occurred during main trading session |
| vwap\_eligible | bit | flag indicating that trade is eligible for VWAP calculation |
| block\_trade | bit | flag indicating that trade is block trade i.e. size of 10K or greater or notional value $200K or greater |

### NBBO Record

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Offset | Size in bytes | Description |
| time | Time | 0 | 8 | time of NBBO |
| bidp | double | 8 | 8 | NBB |
| askp | double | 16 | 8 | NBO |
| bids | integer | 24 | 4 | combined size |
| askst | integer | 30 | 4 | combined size |

### NBBO Price-only Record

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Offset | Size in bytes | Description |
| time | Time | 0 | 8 | time of NBBO |
| bidp | double | 8 | 8 | NBB |
| askp | double | 16 | 8 | NBO |

## Appendix B. Data Formats

|  |  |  |
| --- | --- | --- |
| Field Type | Format | Comments |
| symbol | alphanumeric (18) - up to 18 chars | accepts CTA or UTP symbology |
| timestamp | alphanumeric (36) | ISO format YYYY-mm-dd'T'HH:MM:SS.UUUUUU e.g. "2020-09-01T09:30:00" or "2020-09-01T09:30:00.123456" |
| date | alphanumeric (12) | YYYY-MM-DD or YYYYMMDD |
| time | alphanumeric (20) | time format: HH:MM:SS.UUUUUU or HH:MM:SS e.g. "09:30:00" or "09:30:00.123456" |
| side | alphanumeric (6) | ‘B’,’S’, ‘SS’, ‘BUY’, ‘SELL’, ‘SHORT’  case insensitive recommended values: ‘B’ or ‘S’ or empty string for None |

## Appendix C. VWAP Flavors

Placeholder for description of trade classification and how it is used VWAP calculation.

## Appendix D. Markouts

Markouts is a series of measurements over different time frames around a market event, which widely used in performance analysis. e.g. measuring of market impact may use series of NBBO snapshots taking right after order execution

|  |  |  |  |
| --- | --- | --- | --- |
| Duration Type | Units | Format | Comments |
| Time | microseconds | us, usec | values can be negative, in which case calculation is performed over period ending with reference time |
| milliseconds | ms, msec |
| seconds | s, sec |
| minutes | m, min |
| hours | h, hr |
| Ticks | ticks | t, ticks |

## Appendix E. C++ Build Dependencies

List of commands for installing and building necessary software component 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

## Appendix F. Build Process

Script to build required binaries uses standard CMake utility

#!/bin/bash  
source /opt/rh/devtoolset-8/enable  
scripts="$( cd "$( dirname "${BASH\_SOURCE[0]}" )" >/dev/null 2>&1 && pwd )"  
taq\_proc=$(dirname $scripts)  
build=$(printf "%s/build" $taq\_proc)  
bin=$(printf "%s/bin" $taq\_proc)

if [ ! -d $build ]; then  
 mkdir $build  
fi

if [ ! -d $bin ]; then  
 mkdir $bin  
fi

cd $build  
cmake -DCMAKE\_BUILD\_TYPE=Debug ..  
make -j2

cp -p taq-prep/taq-prep ../bin  
cp -p taq-ctrl/taq-ctrl ../bin  
cp -p tick-calc/tick-calc ../bin  
cp -p taq-py/libtaqpy.so ../bin/taqpy.so  
echo $PATH|sed 's/:/\n/g'|grep $bin 2>&1 > /dev/null

if [ "$?" != "0" ]; then  
 printf "\e[31m \e[5m NB! \e[25m \e[39m Please add %s to PATH \n" $bin  
fi

## Appendix G. Installation Process

Including path to binaries into PATH and PYTHONPATH environment settings is sufficient for successful execution of TDP processes.

scripts="$( cd "$( dirname "${BASH\_SOURCE[0]}" )" >/dev/null 2>&1 && pwd )"  
taq\_proc=$(dirname $scripts)  
bin=$(printf "%s/bin" $taq\_proc)

toolbox=/usr/local/astral

if [ ! -d $toolbox ]; then  
 sudo mkdir $toolbox  
 sudo chown $USER $toolbox  
 sudo chgrp $USER $toolbox  
 printf "Created %s\n" $toolbox  
 ls -ld $toolbox  
fi

if [ ! -d $toolbox/bin ]; then  
 mkdir $toolbox/bin  
fi

cp -p $taq\_proc/bin/taq-prep $toolbox/bin/  
cp -p $taq\_proc/bin/taq-ctrl $toolbox/bin/  
cp -p $taq\_proc/bin/tick-calc $toolbox/bin/  
cp -p $taq\_proc/bin/taqpy.so $toolbox/bin/  
source $taq\_proc/scripts/env.sh