Hanweck Historical Data

API Programmer's Guide

Abstract

This document provides information on business descriptions, programming interfaces and protocols for connecting client applications to Hanweck Historical Data

While reasonable care has been taken in the preparation of this document to provide details that are accurate and not misleading at the time of issuance, this document is distributed to you solely on an "as is" basis. No representations or warranties are made regarding the information contained herein, whether express or implied, including without limitation any implied warranty of merchantability or fitness for a particular purpose or any warranty with respect to the accuracy, correctness, quality, completeness or timeliness of such information. Hanweck Associates, LLC and its affiliates shall have no liability of any kind whatsoever to any third parties in connection with this document and the information contained herein. This document contains Hanweck Associates, LLC's confidential information. It is provided for general informational purposes only, and it may be used solely for your internal business purposes. You may not redistribute this document to anyone outside of your organization. All descriptions, examples, and calculations contained herein are for illustrative purposes only. Unauthorized reproduction or use of any content of this document or the subject matter hereof, including, but not limited to, trademarks, text and pictures, is strictly prohibited. Borrow Intensity™, Volera®, VoleraSpace™, "Real-time risk analytics" and the Hanweck Associates, LLC logo are trademarks and service marks of Hanweck Associates, LLC or its affiliates and may not be used for any purpose.

Table of Contents

Tables

1 About This Document

This document is for programmers, analysts, and IT managers who are developing applications that will use data from Hanweck Historical Data. It covers the general business behavior of the Hanweck Historical Database and the technology standards and techniques employed to provide this service.

2 Overview of Hanweck Historical Data

Hanweck Historical Data is a fully managed historical database of US equities level 1 data, full OPRA tick data and pre-computed derived data including implied volatilities, historical volatilities and Greeks. Hanweck Historical Data also provides extensive corporate-action histories, including splits, dividends and symbol changes, which subscribers can choose to apply. This includes point-in-time options security masters, including OPRA roots, OPRA Symbology Initiative (OSI) symbols, underlying, chains, strikes, expiration dates, strike multipliers and deliverable units/baskets.

2.1 Related Documents

The following are additional documents related to Hanweck Historical Data which detail the available content.

Table 1: Hanweck Historical Data Related Documentation

Document	Description	Location
Hanweck Historical Data Tick Data Content Users' Guide	This document details the Hanweck Historical Data content available for: • Tick Pricing • Tick Analytics Market coverage: • OPRA Trades, Quotes, BBO and End-of-Day Summary and Administrative Messages • U.S. Level 1 Equity Trades, Quotes, and BBO • U.S. Implied Volatilities and Greeks for OPRA universe • Cboe Futures Exchange -CFE Trades, Quotes, and BBO • CME Trades, Quotes, and BBO	Available on Request

Document	Description	Location
Hanweck Historical Data Corporate Actions and End of Day Pricing Content Users' Guide	This document details the Hanweck Historical Data content available for: • Corporate Actions and Reference Data • End of Day Pricing • End of Day Analytics	Available on Request

2.2 Hours of Operation

Hanweck Historical Data runs hot/hot at redundant data centers and is available 24 hours per day, seven days per week. During hours of scheduled maintenance, at least one site will be available. There are both real time intraday data sets and end of day data sets.

3 Direct Access

Hanweck Historical Data can be accessed directly via:

- The Internet
- Direct Connect
- Cross Connect

This robust data solution is hosted in redundant datacenters and offers 365x24 availability for full tick or time interval back-testing of models and algorithms, pre/post trade analysis, charting, time and sales requests, and a vast selection of pre-defined filters to minimize the need for custom development.
A Web browser interface and easy-to-use C/C++/C#/Java APIs are also available.

4 API Programmers Guide

4.1 Introduction

The Hanweck Historical Data API is an application level interface that allows application developers to query the database and retrieve data from remote client applications. Performance is an overriding factor in the design of Hanweck Historical Data and the Hanweck Historical Data API, so that data are returned as quickly as possible.

Client programs can be written in different languages and the API provides the necessary binding to link with the underlying library.

Currently, the API supports different language bindings, they are:

- Java
- C/C++
- C#

This section describes the Hanweck Historical Data API in more detail, and provides a high-level description of the classes and methods involved in using the API. The Hanweck Historical Data API is modeled after the ODBC framework, and should be familiar to users of ODBC and its variants (e.g., JDBC).

4.2 General Concepts

4.2.1 Symbol List

Symbols are entered as a string symbol, an OSI Ticker or OPRA Ticker.

4.2.2 Time Range

A time range for each query is specified in yyyyMMdd:HH:MM:ss.SSS format. For example “20110831:09:30:00.000” equals August 31, 2011 at 9:30am ET.

4.2.3 Exchange

A query can specify whether to include data from all exchanges or just data from a specific exchange or set of exchanges.

4.2.4 Execute / Record Set ODBC

After all parameters have been set, the results of the query are executed and displayed in tabular format.

4.2.5 Database Queries

This section provides a description of each available type of query. Query types and available input parameters are defined by each query’s respective static class.
Corporate-action histories and reference data is applied in a number of ways when querying data from Hanweck Historical Data. Symbol changes are mapped and Hanweck Historical Data follows the contract symbol from the “as of” date through any changes that occurred during the queried time interval. Relisted symbols can also be filtered out. Hanweck Historical Data can also apply equity adjustments which take into account stock-splits, dividends paid, etc. When requested, Hanweck Historical Data will use corporate action history data to adjust the price of each security to be representative of the price as of the requested "as of" date.
The available query types include:

- OHLC Bars
- Snapshots
- Replay (Tick by Tick)
- Time and Sales

4.2.6 OHLC Bars

Open high low close bars are available for OPRA and US Level 1 data with fully configurable formatting. The user has the ability to query from a list of symbols (option and underlying) or optionally choose to query an entire option chain for each inputted underlying symbol. For each query, the user can chose trade, quote, or BBO data series to construct bars with trade volume optionally included. The user can specify composite data or provide a list of exchanges be included in the returned data. Bucket intervals and each of open, high, low, and close values are fully configurable through the use of the API's input parameters.

4.2.7 Snapshots

Snapshots are available for the full range of data, with down to the millisecond precision. They can be queried for either a list of symbols or a list of option chains. The user has the option of specifying a maximum time interval in seconds for the server to search back for the latest quotes and trades.

4.2.8 Ticks (Replay)

Tick queries give the user the ability to retrieve raw tick and trade data for an inputted list of symbols or option chains. The user also has the ability to filter ticks by exchange.

4.3 Sample Code

4.3.1 Get Connection Pool

A class is first created and then a new connection pool is established for querying Hanweck Historical Data's database. User name, password, and Hanweck Historical Data URL are entered.

```

PhDConnectionPool connPool = new PhDConnectionPool(username, password, phdUrl);

```

The PhDConnection class manages the underlying communication details between the client application and the PhD server. At any time, a PhDConnection can have only one pending request and response. (An exception is the "cancel" request, which the client can send after an "execute query" request.) Multiple PhDConnections can be established for concurrent requests.

The PhDConnectionPool class manages a pool of PhDConnections, reusing available connections whenever possible. Once constructed the user has the ability to get a PhDConnection from the PhDConnectionPool rather than constructing one directly. This connection should then be returned to the pool using the returnConnection() method. Returning the connection allows the PhDConnectionPool to reuse connections without requiring the overhead of connection time connecting to the server.

4.3.2 OHLC Bars Sample Code

A bars query is used to generate Open/High/Low/Close bars on time series of ticks. The bars are fully configurable in that the user can decide what tick values (bid/ask/mid) to use for each of the Open, High, Low, and Close values. Optionally trade volume can also be included in the resulting time series. For this query the user can specify BBO, Quote, or Trade tick types while running the query for a list of underlyers and options or a list of option chains.

The sample code below shows an example of an equity bar request on BBO quotes for Apple from August 31 2011 9:30am through 4:00pm in 10 minute intervals across all exchanges, using the mid value and including trade volume information. (Please see Appendix A: for more details regarding input parameters and output fields)

Example Input

Symbol	AAPL
Start Date	2011/08/31 09:30:00.000
End Date	2011/08/31 16:00:00.000
Bucket Interval	600
Type	BBO
Include Volume	1
Bar Type	MMMM

Example Output (one tick shown)

Symbol	AAPL
Time	2011/08/31 10:30:00.000
Open	421.2700000
High	421.5600000
Low	421.0000000
Close	421.4000000

Volume	81910.0000000
Symbol Name	USLEVEL1::AAPL

```
public int getOHLCBars(ArrayList<PhDSymbol> symbols, Date sDate, Date eDate, String type, int seconds, int volume,
boolean print, Date asof, boolean ischain)
 throws PhDException {
```

The parameters of the function are:

symbols: An ArrayList<PhDSymbol> of symbols. Example value: “AAPL”

sDate: Start date. The first bar will have timestamp "seconds" seconds after this time. Example value: “20110831093000000”

eDate: End date. The final bar will be less than or equal to this time. Example value: “20110831160000000”

type: The type of Bars as defined in com.hanweck.phd.api.request.api.RequestOHLCBars. Example value: “REQUEST_TYPE_BBO”

seconds: The width of each bar in seconds. Example value: “600”

volume: Indicator of whether or not to include trade volume in the result set. Example value: “1”

print: Flag whether or not to printout the query results. Example value “True”

asof: The asof date for the provided symbols. Example value: “20110831093000000”

ischain: Indicator saying whether or not to query the whole chain for each inputted symbol. Note that if this is set, the query will expect all symbols to be underlyers. Example value: “False”

A PhDStatement encapsulates a user request. It must be constructed by calling the appropriate factory (e.g., getStatement) method in the PhDConnection class. Then a PhDParamMap holds the user’s query request and input parameters, and thus completely defines a particular query.

```
            PhDConnection conn = connPool.getConnection();
            PhDStatement stmt = conn.getStatement();

            PhDParamMap inParams = new PhDParamMap(symbols, sDate, eDate,
RequestOHLCBars.RequestQueryName);
            inParams.setAsOf(asof);

            inParams.addString(RequestOHLCBars.RequestOHLCType, "MMMM");
            inParams.addInt(RequestOHLCBars.RequestBucketIntervals, seconds);
            inParams.addInt(RequestOHLCBars.RequestVolume, volume);

            if(ischain)
                inParams.addInt(RequestOHLCBars.RequestChain, 1);

```

```
20              if (type != null)
21                  inParams.addString(RequestOHLCBars.RequestType, type);
```

In line 13, RequestOHLCType has been set to “MMMM” to use all mid prices.

A PhDResultSet is returned from the Hanweck Historical Data server when a query executes.

```
22              int row = 0;
23              PhDResultSet rs = null;
24              try {
25                  rs = stmt.executeQuery(inParams);
26
27                  int colCnt = rs.getColumnCount();
28                  if(print){
29                      System.out.println("# of columns: " + colCnt);
30                  }
31
32                  // retrieve meta data
33                  ColInfo[] metaData = rs.getMetaData();
34                  if(print){
35                      for (int i = 0; i < metaData.length; i++) {
36                          System.out.println("column: " + i + ", name: " + metaData[i].name + ", type: "
37      + metaData[i].dataType);
38                      }
39                  }
```

It contains the tabular results of the query.

```
40                  while (rs.next()) {
41                      String symbol = rs.getString(RequestOHLCBars.ReturnSymbol);
42                      Date ts = rs.getDate(RequestOHLCBars.ReturnTimestamp);
43
44                      double open = rs.getDouble(RequestOHLCBars.ReturnOpen);
45                      double high = rs.getDouble(RequestOHLCBars.ReturnHigh);
46                      double low = rs.getDouble(RequestOHLCBars.ReturnLow);
47                      double close = rs.getDouble(RequestOHLCBars.ReturnClose);
48                      double Volume = rs.getDouble(RequestOHLCBars.ReturnVOLUME);
```

The results are then displayed in the various fields.

```
49                      if(print){
50                          System.out.println("row data: " + (row) + ", " + symbol + ", " +
51      dateFormatyyyyMMddHHmmssOutput.format(ts) +
52                                  ", " + open +
53                                  ", " + high +
54                                  ", " + low +
55                                  ", " + close +
56                                  ", "+ Volume);
```

```
            }
            row++;
        } // end while
    }
    catch (PhDException e) {
        //e.printStackTrace();
        throw new PhDException(e.getMessage());
    }
```

The connection to the server is then closed.

```
    finally {
        if (rs != null) rs.close();
        if (stmt != null) stmt.close();
        connPool.returnConnection(conn);
    }
    return row;
}
```

4.3.3 Snapshots Sample Code

The snapshot query can be used to obtain a complete picture of the market at a particular time. The query can be run for either a list of underlyer tickers and option tickers or for a list of underlyer tickers with the "Request Chain" option set to query entire option chains. When running a snapshot query, the start date represents the farthest back Hanweck Historical Data will search to get the current snapshot.

The sample code below shows an example of a query for a snapshot of Apple best bids and offers at August 31 2011 4:00pm across all exchanges. (Please see Appendix A for more details regarding input parameters and output fields).

Example Input

Symbol	AAPL
Start Date	2011/08/31 09:30:00.000
End Date	2011/08/31 16:00:00.000
Request Type	BBO
Request Chain	0

Example Output (One strike)

Symbol	AAPL
Timestamp	2011/08/31 16:00:00.000

Bid	384.81
Bid Size	1
Bid Exchange	Q
Ask	384.84
Ask Size	16
Ask Exchange	Q
BBO Timestamp	2011/08/31 15:59:59.983

```
73

74      public int getSnapshot(ArrayList<PhDSymbol> symbols, Date sDate, Date eDate, boolean print, Date asof, boolean
75      ischain) throws PhDException {
```

The parameters of the function are:

symbols: An ArrayList<PhDSymbol> of symbols. Example value “AAPL”

sDate: Start date. This is the time of the snapshot. Example value: “20110831093000000” **eDate**: End date. This is the time of the snapshot. Example value: “20110831160000000”

print: Flag whether or not to printout the query results. Example value “true”

asof: The asof date for the provided symbols. Example value: “20110831160000000”

ischain: Indicator saying whether or not to query the whole chain for each inputted symbol. Note that if this is set, the query will expect all symbols to be underlyers. Example value: “false”

A PhDStatement encapsulates a user request. It must be constructed by calling the appropriate factory (e.g., getStatement) method in the PhDConnection class. Then a PhDParamMap holds the user’s query request and input parameters, and thus completely defines a particular query.

```
76                      PhDConnection conn = connPool.getConnection();
77                      PhDStatement stmt = conn.getStatement();
78
79                      PhDParamMap inParams = new PhDParamMap(symbols, sDate, eDate,
80      RequestSnapshot.RequestQueryName);
81                      inParams.setAsOf(asof);
82                      inParams.addString(RequestSnapshot.RequestType, RequestSnapshot.REQUEST_TYPE_BBO);//query
83      only BBO
84
85                      if(ischain)
86                              inParams.addInt(RequestSnapshot.RequestChain, 1);
```

In line 88, REQUEST_TYPE_BBO sets the query type to the best bids and offers.

A PhDResultSet is returned from the PhD server when a query executes.

```
        int row = 0;
        PhDResultSet rs = null;
        try {
            rs = stmt.executeQuery(inParams);

            int colCnt = rs.getColumnCount();
            if(print)
                System.out.println("# of columns: " + colCnt);

            // retrieve meta data
            ColInfo[] metaData = rs.getMetaData();
            if(print){
                for (int i = 0; i < metaData.length; i++) {
                    System.out.println("column: " + i + ", name: " + metaData[i].name + ", type: "
+ metaData[i].dataType);
                }
            }
```

It contains the tabular results of the query.

```
            while (rs.next()) {
                String symbol = rs.getString(RequestSnapshot.ReturnSymbol);
                Date ts = rs.getDate(RequestSnapshot.ReturnTimestamp);

                double bid = rs.getDouble(RequestSnapshot.ReturnBid);
                double ask = rs.getDouble(RequestSnapshot.ReturnAsk);
```

The results are then displayed in the various fields.

```
            } // end while
        }
        catch (PhDException e) {
            throw new PhDException(e.getMessage());
        }
```

The connection to the server is then closed.

```
        finally {
            if (rs != null) rs.close();
            if (stmt != null) stmt.close();
            connPool.returnConnection(conn);
        }
        return row;
    }
```

4.3.4 Ticks (Replay) Sample Code

The ticks query is used to retrieve raw quote ticks and trade data from the tick server. The user can specify all exchanges or provide a list of exchanges for which to view quotes and trades. The query can be run for either a list of underlyers and options or for a given list of option chains.
The sample code below shows an example of a query for Apple BBO ticks from August 31, 2011 9:30am through 4:00pm across all exchanges. (Please see Appendix A: for more details regarding input parameters and output fields).

Example Input

Symbol	AAPL
Start Date	2011/08/31 09:30:00.000
End Date	2011/08/31 10:00:00.000
Request Type	BBO

Example Output (one tick)

Symbol	AAPL
Time	2011/08/31 09:30:08.160
BID_PRICE	395.000000
BID_SIZE	5
BID_EXCHANGE	C
ASK_PRICE	396.000000
ASK_SIZE	1
ASK_EXCAHNGE	C
COND	R
SOURCE	U

```
public int getTicks(ArrayList<PhDSymbol> symbols, Date sDate, Date eDate, AbstractList<String> exchangesL1,
AbstractList<String> exchangesOPRA, int composite, boolean print, Date asof, boolean ischain) throws PhDException {
```

The parameters of the function are:

symbols: An ArrayList<PhDSymbol> of symbols. Example value: “AAPL”

sDate: Start date. Example value: “20110831093000000”

eDate: End date. Example value: “20110831160000000”

exchangesL1: A list of level 1 exchanges to include in the results. Example value: “exchanges”

exchangesOPRA: A list of OPRA exchanges to include in the results. Example value: “null”**print**: Flag whether or not to printout the query results. Example value: “True”

asof: The as of date for the provided symbols. Example value: “20110831093000000”

ischain: Indicator saying whether or not to query the whole chain for each inputted symbol. Note that if this is set, the query will expect all symbols to be underlyers. Example value: “False”

A PhDStatement encapsulates a user request. It must be constructed by calling the appropriate factory (e.g., getStatement) method in the PhDConnection class. Then a PhDParamMap holds the user’s query request and input parameters, and thus completely defines a particular query.

```
		PhDConnection conn = connPool.getConnection();
		PhDStatement stmt = conn.getStatement();

				PhDParamMap inParams = new PhDParamMap(symbols, sDate, eDate,
		RequestTicks.RequestQueryName);
		inParams.setAsOf(asof);
		inParams.addInt(RequestTicks.RequestType, RequestTicks.REQUEST_TYPE_BBO);

		if(ischain)
			inParams.addInt(RequestTicks.RequestChain, 1);

		if (exchangesL1 != null) {
			inParams.addArrayString(RequestTicks.RequestExchangesUSLevel1, exchangesL1);
		}
		if (exchangesOPRA != null) {
			inParams.addArrayString(RequestTicks.RequestExchangesOPRA, exchangesOPRA);
		}
```

A PhDResultSet is returned from the PhD server when a query executes.

```
		int row = 0;
		PhDResultSet rs = null;
		try {
			rs = stmt.executeQuery(inParams);

			int colCnt = rs.getColumnCount();
			if(print)
				System.out.println("# of columns: " + colCnt);

			// retrieve meta data
			ColInfo[] metaData = rs.getMetaData();
```

```
            if(print){
                for (int i = 0; i < metaData.length; i++) {
                    System.out.println("column: " + i + ", name: " + metaData[i].name + ", type: " +
metaData[i].dataType);
                }
            }
```

It contains the tabular results of the query.

```
            while (rs.next()) {
                Date ts = rs.getDate(RequestTicks.ReturnTimestamp);
                String symbol = rs.getString(RequestTicks.ReturnSymbol);
                double bidPrice = rs.getDouble(RequestTicks.ReturnBid);
                int bidSize = rs.getInt(RequestTicks.ReturnBidSize);
                String bidExchange = rs.getString(RequestTicks.ReturnBidExchange);
                double askPrice = rs.getDouble(RequestTicks.ReturnAsk);
                int askSize = rs.getInt(RequestTicks.ReturnAskSize);
                String askExchange = rs.getString(RequestTicks.ReturnAskExchange);
                String cond = rs.getString(RequestTicks.ReturnCondCode);
                if(print){
                    System.out.println("row data: " + (row) + ", " + symbol + ", " +
dateFormatyyyyMMddHHmmssOutput.format(ts) +
                            ", " + bidPrice + ", " + bidSize + ", " + bidExchange +
                            ", " + askPrice + ", " + askSize + ", " + askExchange +
                            ", " + cond);
                }
                row++;
            } // end while
            }catch (PhDException e) {
                throw new PhDException(e.getMessage());
            }
```

The connection to the server is then closed.

```
        finally {
            if (rs != null) rs.close();
            if (stmt != null) stmt.close();
            connPool.returnConnection(conn);
        }
        return row;
    }
```

5 Appendix A

5.1 Request / Response Reference Guide

Table 2: Parameters to Request Object

Key	Type	Optional	Default	Description
Symbols	**ArrayList**	**N**		**Setup in PhdParamMap object**
Start Date	Date	N		Setup in PhdParamMap object
End Date	Date	N		Setup in PhdParamMap object
Request Name	String	N		Setup in PhdParamMap object 1) Ticks: tick request 2) Snapshot: snapshot request
SetApplyTimesDaily	Boolean	Y	False	Setup in PhdParamMap object When set the query will be run between the start and end time for each day between startDate and endDate.
Num Cores	Integer	Y	1	Setup in PhdParamMap object Sets the number of CPU cores on which to run the query. This can be set to no more than the user's number of licensed cores
Batch Size	Integer	Y	0	Setup in PhdParamMap object Sets the number of symbols to query in each batch. Default is 0, which will query all symbols in one batch. If set, results will be returned in chunks of batchsize symbols

Table 3: Generic/Common Input Parameters

Key	Type	Optional	Default	Description
RequestChain	Integer (0 or 1)	Y	0	Signals to PHD that the entire option chain should be queried When set to 1, the entire option chain as of the 'asof' date will be queried for each inputted underlying symbol
RequestIgnoreAsOf	Integer (0 or 1)	Y	0	When set to 1, inputted symbols will be used as is for the entire requested date span. Setting this to 1 should give a minor performance boost which can be utilized when symbols are known to not change over the queried date interval
RequestFilterReListed	Integer (0 or1)	Y	1	When set to 0, newly listed options with repeated OSI tickers will be returned alongside the requested options. By default these will be filtered out and the only options returned are contracts which had the inputted symbol on the AsOf date
RequestTradingDays	Integer (0 or 1)	Y	0	When set to 1, only trading days will be queried. To limit this to trading hours only the user should set start date and end date to trading hours and call 'setApplyTimesDaily' on the inputted PhdParamMap object in addtion to setting this parameter

Key	Type	Optional	Default	Description
RequestEquityFactorAdjust	Integer (0 or 1)	Y	0	When set to 1, equity prices will be returned adjusted to the as of date. By default prices will be returned as is, meaning as they were received from the exchange without any adjustment for corporate actions

Table 4: OHLC Bars Request Input

Key	Type	Optional	Default	Description
RequestType	String	Y	REQUEST_TYPE_TRADE_BAR	Determines from which type of ticks OHLC bars will be built Possible Values: REQUEST_TYPE_TRADE_BAR REQUEST_TYPE_QUOTE_BAR REQUEST_TYPE_BBO_BAR
RequestBucketIntervals	Integer	Y	300	The width in seconds of each bar in the result set
RequestExchangesUnderlyers	ArrayString	Y		A list of USLevel1 exchange codes from which to get ticks. When set only ticks from these exchanges will be returned. Example: A,C,D
RequestExchangesOptions	ArrayString	Y		A list of OPRA exchange codes from which to get ticks. When set only ticks from these exchanges will be returned. Example: A,C,D
RequestConditionsUnderlyersQuote	ArrayString	Y		A list of USLevel1 quote condition codes from which to get ticks. When set only ticks with the condition codes will be returned. Example: A,C,D
RequestConditionsUnderlyersTrade	ArrayString	Y		A list of USLevel1 trade condition codes from which to get ticks. When set only ticks with the condition codes will be returned. Example: A,C,D
RequestConditionsOptionsQuote	ArrayString	Y		A list of OPRA quote condition codes from which to get ticks. When set only ticks with the condition codes will be returned. Example: A,C,D
RequestConditionsOptionsTrade	ArrayString	Y		A list of OPRA trade condition codes from which to get ticks. When set only ticks with the condition codes will be returned. Example: A,C,D

Key	Type	Optional	Default	Description
ReqeustVolume	Integer (0 or 1)	Y	1	Set to 1 to get trade volume data. Note that for a Trade type request volume will always be returned
RequestOHLCType	String	Y	MBAM	To specify the O.H.L.C (Open, High, Low, Close) attributes Allowable values are: B: to use bid M: to use mid A: to use Ask For example, String 'MBAM' calculate OHLC with Open(Mid), High(Bid), Low(Ask), and Close(Mid) By default, any unrecognizable character will default to 'Mid'

Table 5: OHLC Bars Return Output

Key	Type	Optional	Default	Description
ReturnSymbol	String	N		
ReturnTimestamp	Date	N		
ReturnClose	Double	N		Close value of OHLC bar
ReturnHigh	Double	N		High value of OHLC bar
ReturnLow	Double	N		Low value of OHLC bar
ReturnOpen	Double	N		Open value of OHLC bar
ReturnVolume	Double	Y		Trade volume on the OHLC interval

Table 6: Snapshot Request Input

Key	Type	Optional	Default	Description
RequestLookBack	Integer	Y	5,200	Longest amount of time to look back for latest tick
RequestConditionsUnderlyersQuote	ArrayString	Y		A list of USLevel1 quote condition codes from which to get ticks. When set only ticks with the condition codes will be returned. Example: A,C,D
RequestConditionsUnderlyersTrade	ArrayString	Y		A list of USLevel1 trade condition codes from which to get ticks. When set only ticks with the condition codes will be returned. Example: A,C,D
RequestConditionsOptionsQuote	ArrayString	Y		A list of OPRA quote condition codes from which to get ticks. When set only ticks with the condition codes will be returned. Example: A,C,D
RequestConditionsOptionsTrade	ArrayString	Y		A list of OPRA trade condition codes from which to get ticks. When set only ticks with the

Key	Type	Optional	Default	Description
				condition codes will be returned. Example: A,C,D
RequestType	String	Y	REQUEST_TYPE_BBO	A list to choose from: REQUEST_TYPE_BBO: to signify a request for the 'BBO' tick type REQUEST_TYPE_VOLERA: to signify a request for the 'VOLERA' tick type REQUEST_TYPE_BBO_AND_VOLERA: to signify a request for the 'BBO' and 'VOLERA' tick types

Table 7: Snapshot Request Return Output

Key	Type	Optional	Default	Description
ReturnSymbol	String	N		
ReturnTimestamp	Date	N		
ReturnBid	Double	Y		Bid Price
ReturnAsk	Double	Y		Ask Price
ReturnBidAskTime	Date	Y		Time of Bid/Ask Tick
ReturnPrice	Double	Y		Last Trade Price
ReturnTradeTime	Date	Y		Last Trade Tim
ReturnExpiryDate	String	Y		Expiry day of month
ReturnExpiryMonth	String	Y		Expiry month (month code)
ReturnExpiryYear	String	Y		Expiry Year
ReturnStrike	Double	Y		Strike value
ReturnVoleraBid	Double	Y		Bid price used in volera calculation
ReturnVoleraAsk	Double	Y		Ask price used in volera calculation
ReturnVoleraBidTimestamp	Date	Y		Exchange timestamp of the bid tick used in volera calculation
ReturnVoleraAsKTimestamp	Date	Y		Exchange timestamp of the ask tick used in volera calculation
ReturnBidVol	Double	Y		Calculated implied volatility at the bid price
ReturnMidVol	Double	Y		Calculated implied volatility at the mid price. Mid price is the average of the bid and ask
ReturnAskVol	Double	Y		Calculated implied volatility at the ask price
ReturnDelta	Double	Y		Calculated Delta
ReturnGamma	Double	Y		Calculated Gamma
ReturnTheta	Double	Y		Calculated Theta
ReturnVega	Double	Y		Calculated Vega
ReturnRho	Double	Y		Calculated Rho
ReturnVoleraUPrice	Double	Y		Underlyer price used in volatility calculation
ReturnVoleraCalcTime	Double	Y		VOLERA.CALC_TS

Table 8: Ticks Request Input

Key	Type	Optional	Default	Description
RequestType	String	Y	REQUEST_TYPE_BBO	A list to choose from: REQUEST_TYPE_TRADE: to signify a request for the 'Trade' tick type REQUEST_TYPE_QUOTE: to signify a request for the 'Quote' tick type REQUEST_TYPE_BBO: to signify a request for the 'BBO' tick type REQUEST_TYPE_TRADE_AND_QUOTE: to signify a request for both 'Trade' and 'Quote' tick types REQUEST_TYPE_TRADE_AND_BBO: to signify a request for both 'Trade' and 'BBO' tick types REQUEST_TYPE_QUOTE_AND_BBO: to signify a request for both 'Quote' and 'BBO' tick types REQUEST_TYPE_VOLERA: to signify a request for the 'Volera' tick type REQUEST_TRADE_AND_VOLERA to signify a request for both 'Trade' and 'Volera' tick types REQUEST_TYPE_BBO_AND_VOLERA: to signify a request for both 'BBO' and 'Volera' tick types REQUEST_TYPE_QUOTE_AND_VOLERA: to signify a request for both 'Quote' and 'Volera' tick types
RequestExchangesUnderlyers	ArrayString	Y		A list of USLevel1 exchange codes from which to get ticks. When set only ticks from these exchanges will be returned. Example: A,C,D
RequestExchangesOptions	ArrayString	Y		A list of OPRA exchange codes from which to get ticks. When set only ticks from these exchanges will be returned. Example: A,C,D
RequestConditionsUnderlyersQuote	ArrayString	Y		A list of USLevel1 quote condition codes from which to get ticks. When set only ticks with the condition codes will be returned. Example: A,C,D
RequestConditionsUnderlyersTrade	ArrayString	Y		A list of USLevel1 trade condition codes from which to get ticks. When set only ticks with the condition codes will be returned. Example: A,C,D
RequestConditionsOptionsQuote	ArrayString	Y		A list of OPRA quote condition codes from which to get ticks. When set only ticks with the condition codes will be returned. Example: A,C,D
RequestConditionsOptionsTrade	ArrayString	Y		A list of OPRA trade condition codes from which to get ticks. When set only ticks with the

Key	Type	Optional	Default	Description
				condition codes will be returned. Example: A,C,D

Table 9: Ticks Request Return Output

Key	Type	Optional	Default	Description
ReturnSymbol	String	N		
ReturnTimestamp	Date	N		
ReturnBid	Double	Y		
ReturnBidSize	Integer	Y		
ReturnBidExchange	String	Y		
ReturnAsk	Double	Y		
ReturnAskSize	Integer	Y		
ReturnAskExchange	String	Y		
ReturnCondCode	String	Y		Quote condition code
ReturnSourceCode	String	Y		
ReturnTickType	String	N		Whether it is TRD: for trade QTE: for quote VOL: Volera
ReturnTradePrice	Double	Y		
ReturnTradeSize	Integer	Y		
ReturnTradeExchange	String	Y		
ReturnAskTimestamp	Date	Y		
ReturnBidTimestamp	Date	Y		
ReturnMidVol	Double	Y		
ReturnBidVol	Double	Y		
ReturnAskVol	Double	Y		
ReturnDelta	Double	Y		
ReturnEstDelta	Double	Y		Estimated delta
ReturnGamma	Double	Y		
ReturnRho	Double	Y		
ReturnVega	Double	Y		
ReturnTheta	Double	Y		
ReturnUnderlyerPrice	Double	Y		
ReturnUnderlyerTicker	Double	Y		

6 Document Revision Table

Table 10: Document Revision Table

Version	Date	Change	Section	Comments
1.0	Oct 4, 2016	Initial document release with content split into Hanweck Historical Data Content Users' Guide and Hanweck Historical Data API Programmer's Guide.		
1.1	Dec 5, 2016	Updated to styles and reference to Hanweck Historical Data.		
1.2	June 30, 2020	Document Rebrand		

Cboe Hanweck™