FAQ – uTrade Backend API

Document Version : 3.0

Date: Apr 13, 2020

Company Overview

uTrade Solutions is a fintech company providing enterprise software for financial trading, including multi-asset trading platform, algorithms and risk management solutions to financial institutions and their end customers. Our product suite includes the following:

- ✓ **Multi Asset Trading platform:** with full suite application and html5 web-based front ends (Including admin functions, risk management, order management, connectivity to exchanges etc.). It supports trading for all listed products including equities, futures, options, commodities, as well as for non-listed products like FX etc.
- ✓ **Low latency algorithmic trading platform:** Used in exchange co-location environment or in client data centers/cloud for fastest access to markets to execute arbitrage, market making, execution, excel based, quant driven, API based proprietary and various other strategies across all asset classes. It also provides FIX APIs for DMA and Algos access.
- ✓ **Open Source, Risk Management, and Custom Solutions**: We also customize and open source some modules of our technologies.
- ✓ **Hashcove -** uTrade's partner firm Hash cove is driving new age technology solutions around crowd sourced digital platforms.

We have built our products from ground-up with a modular architecture in order to effectively address current and rapidly evolving user needs. We have also filed for 6 patents in India and 1 patent in US/UK to lead innovation in the trading life cycle.

Please watch our video demos at

www.youtube.com/utradesolutions and www.youtube.com/hashcove

uTrade Solutions was recognized as a leading innovative fin-tech start-up by

FinTech Innovation Lab

RED HERRING ASIA WINNER 100

NASSCOM®

CII Confederation of Indian Industry

एसटीपीआई STPI

Contents

1. Preface

The objective of this document is to guide the users/developers to understand uTrade's latency sensitive frontend API interface for coding custom strategies. This document explains the features available in the API 2.0.

2. Scope of Document

The scope of this document is to specify frequently asked questions before or during development of an algo using uTrade's API Version 2.0.

3. Frequently Asked Questions

1. **What packages are required for Development?**
   - For Prop build, it is gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-11) - C++ 9
   - For Insti build, it is gcc (GCC) 4.9.2 20150212 (Red Hat 4.9.2-6) - C++ 11
   - Cmake
   - Boost library
   - uTrade's API include files (Refer http://utrade.github.io)
   - uTrade's HFT setup for development testing
   - Order Wrapper & common files (Refer http://utrade.github.io) - Not mandatory.

2. **Do development server and Production server need to be different?**
   They can be same backend server. Build process generally consumes CPU. So, it is recommended to keep different once the live server performance matters.

3. **How many users can be logged in simultaneously on one server?**
   Recommendation is 5 - 6 users. It is governed by message line supported by the exchange.

4. **How many strategies could be run on one server simultaneously?**
   Our server supports 1000 strategies simultaneously. Limitation is put by the message line supported by the exchange not by our software.

5. **Could you help us code our custom strategy?**
   Sample Strategy code is given along with the API to help new developers on-board quickly. We can help write strategies based on mutual agreement on commercials and available bandwidth.

6. **My strategies are not latency sensitive - Do I necessarily need to have Co-Lo presence to use your API platform?**
   No, it is not required. API 2.0 can be used for both latency-sensitive and non-sensitive strategies.

7. Can I create a dashboard to see the results of various strategies running on each server?

Trader Frontend shall show the strategies run by that trader. Admin Frontend shall show individual and combined Trade Book and Net Positions. Logs of various strategies are generated at backend. With little Linux usage, it can be done at backend easily.

8. If I ask you to code my strategies, will you sign an NDA so you don't market the same to other people?

We can help code strategies if we have a mutual agreement on commercials and available bandwidth. We can sign an NDA to keep the logic only for you.

9. Currently the number of legs is limited to 4. I am assuming this is arbitrary and this can be changed. If yes, how?

Or

How to run more than four legs algo from frontend algo form?

This is fixed at front end, as showing and choosing number of legs/symbols at front-end is not feasible and moreover, it will not look very user friendly. In such cases, it is recommended to use some custom file format at back-end to read symbols. User can pass file name from frontend using text field for same. Symbology for symbols already provided in API documentation.

10. The front end allows only custom inputs. Is there any way to have custom outputs?

On front-end there is no way to do it right now, we plan to do it in next release. On backend, you can dump outputs to files.

11. But more generally, I don't understand why is the frontend needed. It places numerous other constraints on the user. As far I can tell, currently the frontend is primarily being used to declare the instruments that I would be subscribing to. Can I not do that in code - maybe use a custom config file or something?

Yes definitely, everything can be done from backend config file only. Inside the strategy code, you will need to write code to read and parse the config file per your need.

Front-end shall still be required to launch a strategy. You can keep a blank form and just use the Run/Stop buttons.

12. Apart from that I can have a bidirectional socket in my strategy code and communicate to a custom UI anyway with all the inputs and outputs I want to have. Is there a specific reason to have the proprietary frontend?

Yes, you can have a bidirectional socket (or other IPC mechanism) in strategy code to communicate with your custom GUI. Backend strategies are dynamic link libraries which can be loaded in backend by issuing a command from proprietary frontend.

Proprietary frontend is useful for Order-Book, Trade-Book, Net Positions and Market Watch Widgets. Also, it allows you to run inbuilt strategies on the same server and platform as your custom algos.

13. What is Order wrapper?

Order wrapper is helpful to maintain state of an order. All code of this class is exposed. Thus, user can do changes as per their requirement and use case.

14. Can I use Order Wrapper for multi leg orders as well?

Yes. We have different order wrapper to cater multileg order. Refer sample code for more detail.

15. Why do we need to reset order wrapper before placing an order?

Order wrapper manages state of one order at a time. So, for every new order, we need to clear references of old orders managed by order wrapper.

16. Do we need to create separate order wrappers for each leg and order mode?

Yes. It manages state of one order at a time.

17. Why is order wrapper not part of API?

Our API can be run without the use of order wrapper but in that case one need to do state management stuff individually what is being done by order wrapper. We had tried to give full flexibility to user by exposing order wrapper code.

18. Why calling *processConfirmation* of OrderWrapper is mandatory on every confirmation received?

To maintain position and current state of an order, *processConfirmation* need to be called on every confirmation received.

19. What are pre-trade events in NSE?

In case of tbt optimization build, we give pre trade events to API algos. It has Price, qty, mode and depth details as a prediction that order is filled before actual trade tick given by exchange. Strategy can take decision on the basis of same to attain better efficiency.

20. What is the purpose of Instrument class?

Instrument class contains symbol and position specific data. Static data, position data can be referred from instrument object.

21. Do we need to create instruments for each and every symbol which is being used by strategy?

If one wants to place an order and moreover want to receive market data event for a symbol then instrument need to be created for same.

22. Can we create more than one Instrument class objects for same symbol?

Yes. It is allowed to create more than one instrument for same symbol.

23. What is the significance of OrderId structure in general and use of same in OrderWrapper class?

OrderId structure is helpful to identify an order uniquely at API level. One can send multiple orders in same or different symbol by assigning different orderID. We are passing orderID in process confirmations callback as a reference for identification.

24. How to register for market data with HFT?

One can register for market data using *createNewInstrument* call as MktData class object is member of Instrument class or create object of MktData class directly. Benefit of creating Instrument is algo will get callback in *onMarketDataEvent* but its size is more than MktData class.

25. How one can get index feed in API algo?

One can register symbol using *createNewInstrument* call or create object of MktData class as well. Index value can be retrieved from getLastTradePrice method of MktData class.

26. What is the meaning of OHLC events?

HFT maintains open price, high price, low price and close price corresponding to a time interval for every symbol (the maintenance is configurable and so is the time interval).

An API strategy can register for getting callbacks on updates in OHLC using the following interface methods of *sgContext.h*

- Enable OHLC when registering for market data for a symbol using: *reqRegisterMarketData* or *createNewInstrument*
- *reqStartOHLCAlgo* to start listening to OHLC events for strategy
- After registration, user will start getting OHLC timeout event callbacks in virtual method *onOhlcTimeOutEvent*
- User can use *reqQryOHLCQuote* to get the OHLC quote and use it as required

27. Which fields are present in market data: TBT vs Snapshot in NSE?

Snapshot contains market depth (up to level 5), open price, high price, low price, close price, volume, last trade qty, last trade price, last trade time, average trade price, lower/upper price limit, lower/upper trade execution range etc. whereas TBT contains market depth (up to level 10 by default or configurable if required), volume, last trade qty, last trade price, last trade time etc..

28. What is the time to get tbt tick in strategy?

Usually it is under 1 microsecond but it can vary depending upon number of factors like number of strategies running at a time, computation time of running algo, logic of algo (complex or easy), clock speed of system etc...

29. Can we get raw tbt ticks in strategy?

No. Only final updated picture (created using raw tbt data ticks) is accessible in strategy for now.

30. Can I get market data of tbt and snapshot for same symbol in same algo?

Yes. While registering symbol using *createNewInstrument* call back, there are two params which need to pass for market data register. One is *useTbt* and another one is *useSnapshot*. User can set any combination of both. In case user set both as false then by default snapshot will be considered as true. To get market data, there some method calls in base strategy

reqQrySnapshotMarketData, *reqQryTbtMarketData* and *reqQryMarketData* which can be used to get market data.

31. How to differentiate between tbt and snapshot market data in an API algo?

There are request methods in base strategy named r*eqQrySnapshotMarketData*, *reqQryTbtMarketData*, *reqQryMarketData, reqQryUpdateSnapshotMarketData*, *reqQryUpdateTbtMarketData* and *reqQryUpdateMarketData* for same. Object pointer in which user is assigning above mentioned method's output should be different for snapshot and tbt.

32. Can I register for less than available market depth?

Yes. In *createNewInstrument,* there is a parameter depthsize where user can specify depth to process. It shouldn't be less than 1 and more than max depth. For max depth, refer

MarketDepthArraySize in *apiConstants.h*. Same option is available in *MktData* structure as well.

33. Will I get all snapshots / tbt ticks in onMarketDataEvent?

Not necessarily as there may be a bit delay between tick received from exchange and callback scheduled for strategy. So, user will get latest data available when callback scheduled.

34. How to fetch updated market data?

Call to *reqQryUpdateMarketData* will give updated market data to user. Update method of MktData class can also be called for same.

35. *reqQryMarketData* vs *reqQryUpdateMarketData*?

MktData pointer given by *reqQryMarketData* may have old market data but in case *reqQryUpdateMarketData*, system syncs market data with HFT's market data store.

36. Can I opt to run my backend algo with template-based frontend form (automated) and custom frontend form at same time?

Yes. Backend algo is unaware about the client part. It needs params detail which should be from any form. There may be one-time minor changes required to accomplish same which will not take more than 3-4 hours.

37. Should we employ sleep in strategy callbacks?

It is not recommended to put sleep in strategy callbacks as these callbacks is being scheduled by scheduler itself directly. So, consuming critical time of scheduler and main thread slot in one strategy may impact overall performance of the system.

38. What is the difference between *createNewInstrument*, *reqRegisterMarketData* and *new MktData*?

createNewInstrument - Registers symbol and also maintains position of same symbol for orders placed using same. *reqRegisterMarketData* - Registers market data to get callback in onMarketDataEvent.

new MktData - Creates MktData object but to get updated market data, user need to call update method every time. No callback will be received in such cases.

39. What is the minimum time that can be passed to reqTimerEvent method?

It usually depends upon clock speed of system but recommended value for a reasonable system is 20 microseconds.

40. How to send custom data to frontend?

There are some methods in base strategy named *reqQrySendCustomResponse* and *reqQrySendCustomResponseEx* for same. User need to pass column name

and data where one wants to display received data. Multiple rows can be passed using a list. Refer sample code for demonstration.

41. While creating symbol id using symbol name, why DEFAULT needs to be added?

DEFAULT refers broker option. All exchanges which are directly connected with uTrade are under DEFAULT. For exchanges which are connected through a broker like TT, IB etc. required TT, IB respectively. So, there may be a chance that same exchange connected directly and through broker as well. To differentiate same, DEFAULT or other broker option need to be added.

42. How to handle multiple orders? Do I need to keep track of multiple orders myself?

Yes. One need to track order wrappers using order id. Each order wrapper should have different orderId which will help to handle and track multiple orders.

43. What is static data?

Static data class contains all data related to a symbol which is provided by exchange like scrip code, market lot, expiry, strike price, tick size etc.

44. Is stop order supported, and is there a higher-level abstraction for stop loss on complete portfolio?

Individual stop loss order is supported in API but it is not supported on complete portfolio as of now. User can write his/her own logic in API algo to achieve same.

45. Can I register symbols on run time?

Yes. It can be registered at any time as per requirement.

46. Can we build our custom frontend?

Yes. We have an API to build custom frontend form targeting backend strategy known as uTrade Frontend algo API. Moreover, whole proprietary frontend with all features can also be developed using our .Net API. Refer http://utrade.github.io for more detail.

47. What is the significance of CMakeList?

CMakeLists.txt is used to define what binaries need to build using which cpp files, what libraries need to be linked etc.

When creating API strategy, you will need to define the module which will create a shared object file where client logic resides. It creates makefile of project and then using make command, user can compile his project.

48. How do we build our strategy?

First of all download sample algo from http://utrade.github.io and copy in directory where one want to compile.

- Create directory named build and bin inside newly created directory.
- Go to the build directory and run the following command:
- For Debug build: cmake -DCMAKE_BUILD_TYPE="Debug" ..
- For Release build: cmake -DCMAKE_BUILD_TYPE="Release" ..
- run make command (make).
- SO file will be generated inside the bin directory.

49. In total how many types of events we support?

- Market data event
- Polling based timer event
- Pre-trade event (custom)
- Trade tick event
- OHLC data event
- Admin message callback event
- Exchange message callback event
- Order confirmation callback event
- Invoke in-built strategy event

50. How much C++ do I have to know for coding an algo using uTrade API?

Basic C++ and STL is sufficient to code an algo using uTrade's backend API.

51. How much is the time required to able to start coding with your API? In how much time can I code the algo?

Most of the time it depends upon complexity of algo but if one is having basic knowledge of C++ and STL then a sizeable algo like two leg arbitrage can be written within 20-25 days.

52. How do I test my coded algo?

One can load algo with the help of a command from frontend. System will create frontend form based on template passed where user can give input to run multiple portfolios and can test all combinations. We also provide a simulator Which runs with recorded live feed to test the algo. Moreover, command line API client can also be used for same.

53. Is there an API to record market data for quantitative analysis?

There is no specific API for same but one can use existing algo API to record symbol feed in algo as all feed parameters and different feeds (snapshot, TBT) are supported in API.

54. Can you provide API so I can directly call uTrade inbuilt algos like Cash Fut or others?

Yes, we have an invoking API embedded in same backend API which is helpful to call any in-built algo from API algo. One can run single or multiple in-built algos from single API algo.

55. How to configure Algo Id for API strategy?

There is a method provided at API interface as readConfForAlgoid() to configure your Algo's Id with exchange and exchange Segment wise, this method is exposed in sgContext.h.

56. Can I get the Dealer Positions in API strategy?

Yes, there are many methods exposed with different keys to get the positions.

2.1 Dealer Global Position
2.2 Dealer + Symbol Position
2.3 Dealer + Client + Symbol Position
2.4 Client Global Position

2.5 Client + Symbol Position
2.6 Symbol Position

Methods are exposed for Insti build in sgContext.h

57. Can I get the Used deposit of the client and dealer in API strategy?

Yes, there are methods exposed for Client and Dealer Used deposits in API and you can also get Order Limits for the same.

Method exposed for Insti build in sgContext.h.

58. Can I get all the client code mapped with a specific dealer?

Yes, there is a method exposed in sgContext.h where you can get clients mapped with Dealer + exchange + segment.

Method exposed for Insti build in sgContext.h (method name: getClientCodesDealerExchangeSegmentWise)

59. Can I run an interexchange strategy w.r.t to NSE in API strategy?

Yes, for interexchange strategies you can set a field in your strategy which will set 13th digit of location Id with '4' in case of NSE orders.

Method exposed for Insti build in sgContext.h (method name: setStrategyType).

uTrade’s Client presence

info@utradesolutions.com

uTrade Solutions Private Ltd.

2nd Floor, Quark Atrium, A-45,

Industrial Focal Point, Phase 8-B,

Industrial Area, Mohali – 160 071,

Chandigarh, India