Creating new strategy using API version 2.0

Company Overview

uTrade Solutions is a fintech company providing enterprise solutions including Blockchain driven clearing, 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, CFDs, etc.
- ✓ **Low latency algorithmic trading platform:** Used in exchange co-location environment or in client data centres/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 provide FIX APIs for DMA and Algos access
- ✓ **Open Source, Risk Management, and Custom Solutions:** We also customise and open source some modules of our technologies.

- ✓ **Blockchain Solutions:** uTrade is driving new solutions enabled by the latest technologies including Blockchain. We have built Clearing and KYC focused Blockchain platforms. We are also partnering with clients to guide and provide Blockchain Solutions for their desired workflows. Our existing products include:
  - **uClear** – uClear is a Blockchain based clearing solution for real time clearing and settlement. It allows for any exchange-matching engine to clear trades post execution through a private blockchain, across equities and futures, with real time risk management, reporting and other financial transfer instructions.
  - **KYChain** - KYChain is a KYC platform with mobile and web driven interface for clients to upload their documents and share securely with any institutions, primarily for KYC purpose. Institutions can also contribute back to the data of the users based on permissions.

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
In 2013-2014, uTrade Solutions was recognized as a leading innovative fin-tech start-up by:

FinTech Innovation Lab

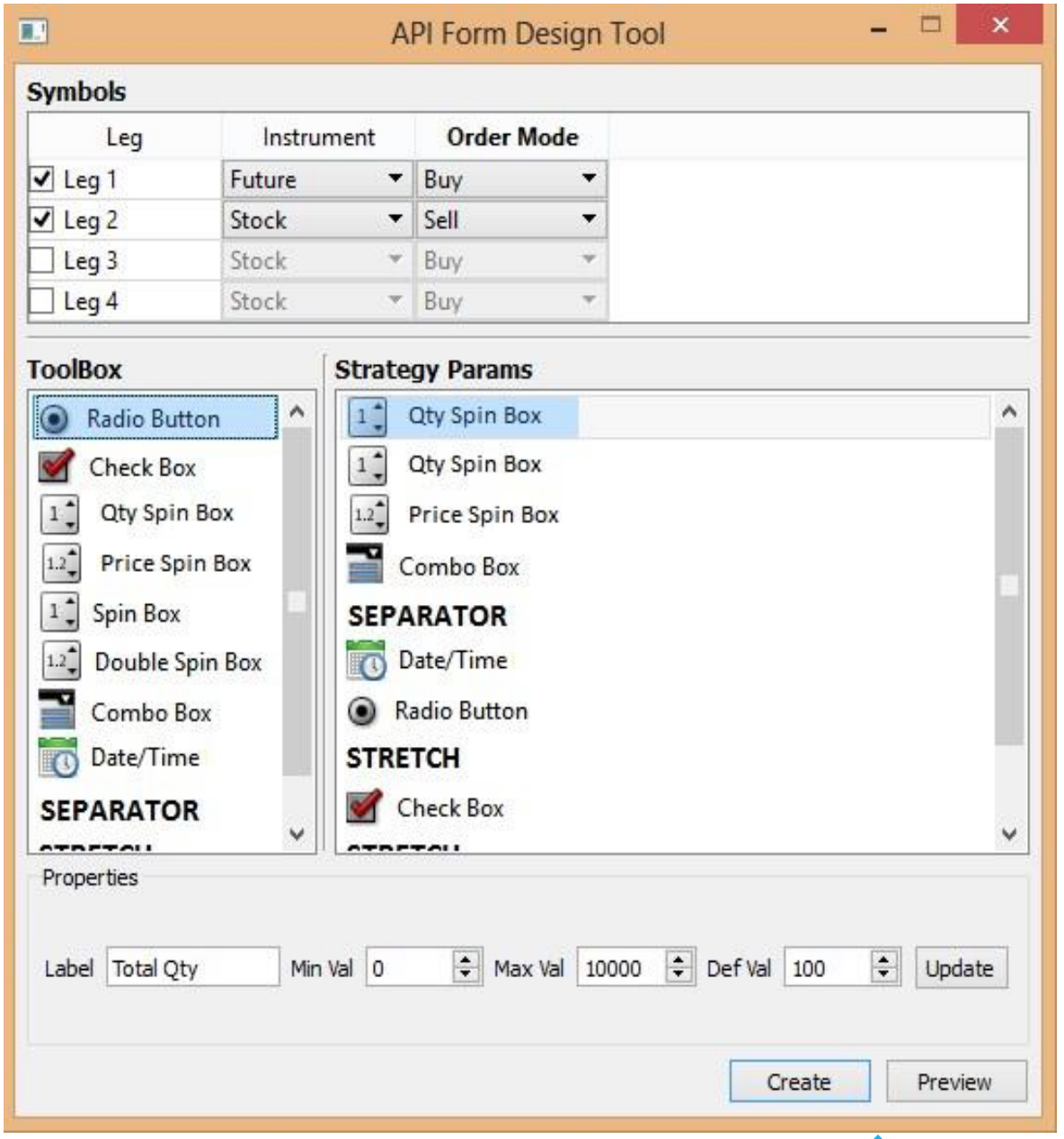
RED HERRING ASIA WINNER 100

NASSCOM®

CII Confederation of Indian Industry

Front-end Design

Design front end using uTrade design toolbox. Choose and model parameters according to your needs. Choose label name wisely as same will be used to get values from backend.



Download front end script and embed in backend strategy code

use linux command xxd -i <parameters_frontfile>,

This will convert text into embeddable code

create a function

```
extern "c"  std::string getFrontEndDesign();
```

to return this design

xxd- i params.txt

```
unsigned char param_txt[] =
{
    0x5b, 0x53, 0x59, 0x4d, 0x42, 0x4f, 0x4c, 0x5d, 0x0a,
    0x53, 0x59, 0x4d, 0x42, 0x4f, 0x4c, 0x20, 0x4c, 0x45, 0x47, 0x31,
    0x4e, 0x54, 0x36, 0x34, 0x0a, 0x4f, 0x72, 0x64, 0x65, 0x72, 0x20,
    0x4d, 0x6f, 0x64, 0x65, 0x20, 0x31, 0x3d, 0x55, 0x43, 0x0a, 0x53,
    0x59, 0x4d, 0x42, 0x4f, 0x4c, 0x20, 0x4c, 0x45, 0x47, 0x32, 0x3d,
    0x55, 0x49, 0x4e, 0x54, 0x36, 0x34, 0x0a, 0x73, 0x74, 0x20, 0x42,
    0x69, 0x64, 0x2f, 0x41, 0x73, 0x6b, 0x3a, 0x42, 0x65, 0x73, 0x74,
    0x20, 0x41, 0x73, 0x6b, 0x2f, 0x0a, 0x53, 0x45, 0x50, 0x41, 0x52,
    0x41, 0x54, 0x4f, 0x52, 0x3d, 0x31, 0x0a, 0x54, 0x69, 0x6d, 0x65,
    0x72, 0x3d, 0x54, 0x52, 0x0a, 0x53, 0x70, 0x72, 0x65, 0x61, 0x64,
    0x20, 0x54, 0x79, 0x70, 0x65, 0x3d, 0x52, 0x41, 0x44, 0x49, 0x4f,
    0x3a, 0x64, 0x75, 0x63, 0x74, 0x3a, 0x41, 0x63, 0x74, 0x75, 0x61,
    0x6c, 0x3a, 0x41, 0x63, 0x74, 0x75, 0x61, 0x6c, 0x0a, 0x53, 0x54,
    0x43, 0x48, 0x3d, 0x31, 0x0a, 0x43, 0x68, 0x6b, 0x20, 0x53, 0x70,
    0x72, 0x65, 0x61, 0x64, 0x3d, 0x42, 0x4f, 0x4f, 0x54, 0x52, 0x45,
    0x54, 0x43, 0x48, 0x3d, 0x31, 0x0a, 0x42, 0x69, 0x64, 0x20, 0x44,
    0x69, 0x66, 0x66, 0x3d, 0x42, 0x4f, 0x0a, 0x5b, 0x4f, 0x54, 0x48,
    0x45, 0x52, 0x5d, 0x20, 0x20, 0x0a

};
unsigned int param_len_text = 425;
```

Corresponding embedded code

```
extern "C"
std::string getFrontEndDesign()
{
        const char param_txt[] =
        {
            0x5b, 0x53, 0x59, 0x4d, 0x42, 0x4f, 0x4c, 0x5d, 0x0a, 0x53,
            0x59, 0x4d, 0x42, 0x4f, 0x4c, 0x20, 0x4c, 0x45, 0x47, 0x31,
            0x3d, 0x55, 0x49, 0x4e, 0x54, 0x36, 0x34, 0x0a, 0x4f, 0x72,
            0x64, 0x65, 0x72, 0x20, 0x4d, 0x6f, 0x64, 0x65, 0x20, 0x31,
            0x3d, 0x55, 0x43, 0x48, 0x41, 0x52, 0x0a, 0x53, 0x59, 0x4d,
            0x42, 0x4f, 0x4c, 0x20, 0x4c, 0x45, 0x47, 0x32, 0x3d, 0x55,
            0x49, 0x4e, 0x54, 0x36, 0x34, 0x0a, 0x4f, 0x72, 0x64, 0x65,
            0x72, 0x20, 0x4d, 0x6f, 0x64, 0x65, 0x20, 0x32, 0x3d, 0x55,
            0x43, 0x48, 0x41, 0x52, 0x0a, 0x23, 0x53, 0x59, 0x4d, 0x42,
            0x4f, 0x4c, 0x20, 0x4c, 0x45, 0x47, 0x33, 0x3d, 0x55, 0x0a
        };
unsigned int param_text_len = 425 ;
return std::string(param_txt,param_txt_len);
}
```

Create getDriver function

create a function

```
extern "C"
void * getDriver(void * params)
```

this function will be the entry point for your strategy. Here you create an object of you strategy class and start your algo

Sample getDriver function

```
extern "C"
void *getDriver(void *params)
{
    API2::StrategyParameters *sgParams =
    API2::StrategyParameters*)params;
    SampleStrategy context(sgParams);
    return context.reqStartAlgo(true); // Market event required
    //return context.reqStartAlgo(false); // Market event not
    required
}
```

Creating Strategy Class

All strategy need to be derived from `API2::SGContext` class This class has predefined functionality to send order and receive market data & virtual functions which user can override.

Sample Strategy Class

```
class SampleStrategy : public API2::SGContext
{
    public :
    Context(API2::StrategyParameters *params);
    //overridden void
    OnCMDModifyStrategy(API2::AbstractUserParams*); private:
    SINGNED_LONG _symbolid;
    //will be set by front end parameters
    API2::SGCommon::InstrumentOrderId *_instrumentOrderId;
    //used to uniquely identify order
    API2::COMMON::Instrument * _instrument;
    //used in market data subscription , sending orders
    // and p&l calculation
};
```

Things to do in Constructor

-> setup log file using Base class constructor

-> read front end parameters

```
SampleStrategy(API2::StrategyParameters *params)):
API2::SGContext(params, "SampleStrategy") // Sample strategy
log file created
{
    API2::UserParams *frontendParams = (API2::UserParams *)
    params->getInfo();
    //Parameters received from frontend
    //reading parameters
    if(frontendParams->getValue("SYMBOL LEG1",_symbolid) !=
    API2::UserParamsError_OK)
    {
        std::cout<<"Error in getting SYMBOL LEG1"<<std::endl;
    }
 }
```

Subscribing Market Order and Setting up Instrument

```
_instrument = createNewInstrument(
//sending by reference will be set internally_symbolId,
//will be used for creating instrument true,
//need to register for feed or not false);
//feed type false for TBT feed or true for snapshot feed
```

Receiving Market Event

if `ReqStartAlgo` funtion is with market event enabled then feed event will be received by overriding

`onMarketDataEvent`

```
void SampleStrategy::onMarketDataEvent(UNSIGNED_LONG symbolid)
 {
    std::cout<<"Symbol id Market Data "
    <<symbolid
    <<std::endl;
     API2::COMMON::MktData *mkData = reqQryUpdateMarketData(symbolid);
     mkData->dump();
 }
```

Sending Orders

```
API2::SingleOrder *order;
API2::DATA_TYPES::RiskStatus riskStatus =
API2::CONSTANTS::RSP_RiskStatus_MAX;
order = createNewOrder(_instrument, <QTY>, <Revealed QTY>,
API2::CONSTANTS::CMD_OrderMode_BUY,
API2::CONSTANTS::CMD_OrderType_LIMIT,
API2::CONSTANTS::CMD_OrderValidity_DAY,
API2::CONSTANTS::CMD_ProductType_DELIVERY,
<PRICE>);_instrumentOrderId = NULL:
if( ! reqNewSingleOrder(riskStatus,_instrument,_order,
_instrumentOrderId))
//_instrumentOrderId is used to uniquely identify an order
{
//error in sending order
//riskstatus variable will have error code store in it , describing
the problem.
}
else
{
//order sent out successfully
}
```

Receiving Confirmations

SGContext class has many virtual functions such as

- onNewConfirmed
- onCanceled
- onReplaced
- onReplaceRejected
- onCancelRejected
- onNewReject
- onFilled
- onPartialFill

they need to be overridden to capture confirmations such as

```
void SampleStrategy::onConfirmed(API2::orderConfirmation
&confirmation, API2::COMMON::InstrumentOrderId *orderId)

{
    if(orderId == _instrumentOrderId)
    {
        //confirmation received from exchange //do something
        according to your logic
    }

}
```

uTrade

info@utradesolutions.com
uTrade Solutions Private Ltd.
4th Floor, Landmark Plaza (F3), Quarkcity SEZ,
A40A, Industrial Area Phase 8 Extension,
Mohali – 160 071,
Chandigarh, India