2/25/2019

NEUTRON

QUICK START GUIDE
NEUTRON API - 4.4

Table of Contents

Overview

Fundamental Interactions Neutron (FIN) offers native message API to its customers for fast and easy integration. Clients can choose between C++, C# and Java API for ultra-low latency connections and market data.

Client applications, which can be desktop or black box based, will communicate with Neutron servers using different types of messages. These messages are made available through the API layer.

FIN native API messages are backward and forward compatible. New messages and new fields can be added to the API but existing messages and fields will never change so users of previous API versions will not be affected when new versions are released. If a user wants to access to new messages or fields, they can simply download a new library.

Most of Neutron native messages and related fields are self-explanatory. They will not be listed here. Questions can be directed to FI support email: support@fundamentalinteractions.com

The following is a condensed version of the FIN Native API document.

Architecture

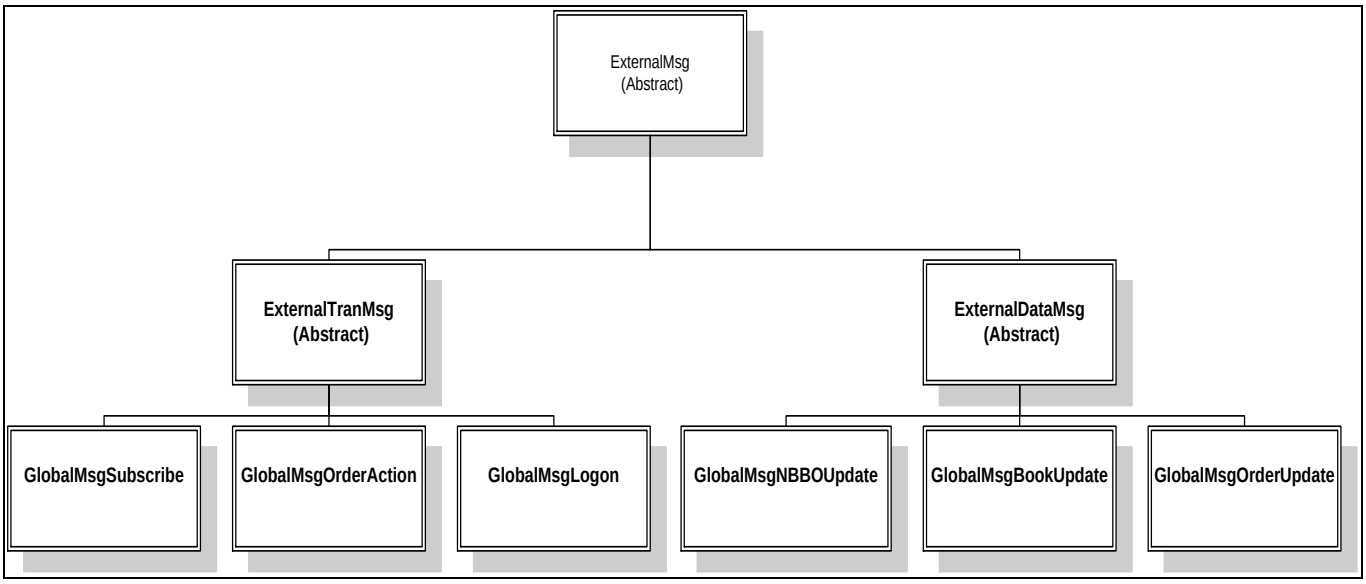
FIN native message API is composed of logical messages. These messages are used to pass information between FIN servers and the client application via TCP. Each message is represented as a data structure. FIN API provides utility to marshal those messages into binary streams for network transmission, or to un-marshal data from the binary streams received.

FIN servers always run in redundant mode. For each module in FIN, there is always a backup module running in standby mode beside primary module.

Upon setup, the client will receive a primary logon address and a secondary logon address along with login credentials. As an operating practice, the client should always try to connect to primary address first, and attempt the secondary address if not successful, using round robin between the two addresses until successful login is accomplished.

Data Types

The hierarchy of FIN native messages is illustrated in the following diagram



Base Class

The base class is ExternalMsg, from which all other classes are derived. From there, messages are divided into two categories, transactional messages (ExternalTranMsg) and subscription messages (ExternalDataMsg).

Transactional Messages

Transactional messages generally have a one to one relationship. The client sends out one request and expects one response. In the example of order entry, the client sends out an order entry request and will receive an order entry response. The response will carry success or failure information.

Subscription Messages

Subscription messages allow subscription to continuous message updates, such as NBBO, book, orders/trades, etc. The client can use GlobalMsgSubscribe to send out requests for subscription and will then receive continuous updates for corresponding messages. If the client wishes discontinue updates, a GlobalMsgUnsubscribe message can be sent and the corresponding updates will stop.

Library Download

The message libraries are available at ftp.finteractions.com, user anonymous@finteractions.com, you need to have a sftp capable client application such as filezilla to access the ftp site.

Symbology

Equities

For equity, FIN uses CMS symbology format and there is a space between root symbol and suffix. For example, for security ZZZ, in security field, the client sets:

ZZZ

For security ZZZ Preferred A, in the security field, the client sets:

ZZZ PRA

Suffix

Security Categorization	CMS Suffix	BATS Suffix
Preferred	PR	-
Preferred Class “A”*	PRA	-A
Preferred Class “B”*	PRB	-B
Class “A”*	A	.A
Class “B”*	B	.B
Preferred when distributed	PRWD	-$
When distributed	WD	$
Warrants	WS	+
Warrants Class “A”*	WSA	+A
Warrants Class “B”*	WSB	+B
Called	CL	*
Class “A” Called*	ACL	.A*
Preferred called	PRCL	-*
Preferred “A” called*	PRACL	-A*
Preferred “A” when issued*	PRAWI	-A#
Emerging Company Marketplace	EC	!
Partial Paid	PP	@
Convertible	CV	%
Convertible called	CVCL	%*
Class Convertible	ACV	.A%
Preferred (class A) Convertible	PRACV	-A%
Preferred (class A) when Distributed	PRAWD	-A$
Rights	RT	^
Units	U	=
When issued	WI	#
Rights when issued	RTWI	^#
Preferred when issued	PRWI	-#

Class "A" when issued*	AWI	.A#
Warrrant when issued	WSWI	+#
TEST symbol	TEST	~

Options

For options, FIN uses explicit option symbology. For example, the ZZZ October 22nd 2010 call option for strike price of $220, the client sets the following in the security field:

ZZZ 20101022C 220.000

Notice that there are always three decimal points in the strike price. If strike price is 220.5, it is be formatted as 220.500

Futures

For Futures and options on futures we use the Exchange Symbol with MaturityMonthYear, Strike Price and PutOrCall clearly identifying the full parameters of the contract.

Foreign Exchange

FX symobology is the ISO standard, like “EUR/USD”, “GBP/USD”

Canada Exchanges

Symbol:COUNTRYCODE

Example like RIM:CA for Canada’s RIM.
If you don’t specify anything country code, default is US, which is what we currently have.

Crypto Currencies

Crypto Currencies symobology is like “BTCUSD”, “BTC/ETH”

API sample usage explanation

Starting with a simple example, assume the client wants to connect to FIN to subscribe for the NBBO, book feed and then to send out an order. For this example the TCP network details will be skipped in order to focus on the message API.

Log In Request

After the TCP socket connection is established a logon request must be sent. First, for the creation of the logon message:

In C++:

```
msggen::GlobalMsgLogon logon;
        logon.init();
```

```
logon.header.firm = firm;
logon.data.userId = userid;
logon.data.version = msggen::Const_msggen::BuildTag;
logon.data.heartbeatInterval = 60;
```

In C#:

```
GlobalMsgLogon logon = new GlobalMsgLogon();
logon.init();
logon.header.firm = firm;
logon.data.userId = userid;
logon.data.version = Const_msggen.BuildTag;
logon.data.heartbeatInterval = 60;
```

In Java:

```
msggen.GlobalMsgLogon logon = new msggen.GlobalMsgLogon();
logon.init();
logon.header.firm = firm;
logon.data.userId = userid;
logon.data.version = msggen.Const_msggen.BuildTag;
logon.data.heartbeatInterval = 60;
```

Data Marshal

Next, the message must be composed into the binary stream for transmission.

In C++:

This function is called:

```
static int msggen::ExternalMsgFactory::compose (msggen::ExternalMsg *msg, char* buffer, int startPos);
```

Outbound messages from the client to the server are generally small, but client should make sure the buffer passed has enough room for the binary stream output - benchmark to a minimum of 30k.

In C#:

```
int msggen.ExternalMsgFactory.compose (msggen.ExternalMsg msg, byte[] buffer, int startPos);
```

Outbound messages from the client to the server are generally small, but client should make sure the buffer passed has enough room for the binary stream output - benchmark to a minimum of 30k.

In Java:

```
public static int msggen.MSGGENFactory.compose (msggen.ExternalMsg msg,
msggen.BByteBuffer buffer);
```

BByteBuffer automatically increases its buffer if there is not enough room.

Then, the client can get the byte[] array from buffer through its buffer field and the length by its getLimit() member function after the flip() member function is called.

Data Unmarshal

After the client application sends a logon message to the server, FIN validates the logon and returns the logon response. In the client application, there is generally a loop to decompose the messages that it receives from the socket, examples are listed below.

The message size from the FIN server to the client varies dramatically depending on what is subscribed for. For example, one GlobalMsgBookUpdate message may contain a few thousand BookData records. Therefore FIN recommends minimum receiving buffer size as 2M.

In C++:

Here is sample program's loop:

```
short receiveMsg (std::vector<msggen::ExternalMsg*> &list)
{
int length = 0;
int outEndPos;
int output = recv (sock, readBuffer+ currentReadPos+bytesRemainRead, BUFSIZE, 0);
if (output < 0)
        return -1;
bool continueProcessing = true;
bytesRemainRead += output;
try
{
        while (continueProcessing)
        {
                int msgLen = msggen::ExternalMsgFactory::getMsgLen (readBuffer,
currentReadPos, bytesRemainRead);
                if (msgLen <= 0 || msgLen > bytesRemainRead)
```

```
                {
                        if (currentReadPos > 0 && bytesRemainRead > 0)
                        {
                                memmove (readBuffer, readBuffer+currentReadPos,
bytesRemainRead);
                        }
                        currentReadPos = 0;
                        return 0;
                }
                msggen::ExternalMsg* result =  msggen::ExternalMsgFactory::decompose
(readBuffer, currentReadPos, outEndPos);
                if (result != NULL)
                        list.push_back(result);
                        currentReadPos = outEndPos;
                        bytesRemainRead -= msgLen;
                }
        }
        catch (msggen::MsgException e)
{
                std::cerr <<"exception "<<e.getError() << std::endl;
                return -1;
        }
        return 0;
}
```

Overriding the global object allocator:

In the C++ library we have the following function:

```
a function ExternalMsg* ExternalMsgFactory:: decomposeMsg (ClientAllocator
*allocator,char* buffer, int startPos, int &endPos) throw(MsgException);
```

A client should override allocatMessage function to return the cached instances.

If the client pass NULL for ClientAllocator in ExternalMsgFactory::decomposeMessage or allocateMessage return NULL, then FIN library will create its own instance and return it.

Where a client can pass a pointer to an ClientAllocator object.

Below is the base Client Allocator:

```
class ClientAllocator
 {
 public:
    ClientAllocator(){}
    virtual ~ClientAllocator(){}
    virtual ExternalMsg* allocatMessage (GlobalMsgType msgType)
    {
        return NULL;
    }
 };
```

In C#:

Here is sample program's loop:

```
{
int length = 0;
int outEndPos;
int output = s.Receive (readBuffer,currentReadPos,MAX_BUFFER_SIZE-
currentReadPos,0);
if (output <= 0)
return -1;
totalBytes += output;
currentReadPos += output;
while (currentReadPos - currentDecompPos > 5)
{
ExternalMsg.getInt(out length, readBuffer, currentDecompPos + 1);
if (length == 0)
throw new MsgException(MsgException.MSG_ERROR_SOCKET_STX, "NO Length");
if (currentReadPos - currentDecompPos < length)
break;
 B ExternalMsg msg = msggen.ExternalMsgFactory.decompose(readBuffer,
currentDecompPos, out outEndPos);
currentDecompPos = outEndPos;
if (msg != null)
list.Add(msg);
      }
if (currentDecompPos == currentReadPos)
{
currentDecompPos = currentReadPos = 0;
}
else
```

```
{
if (currentDecompPos > readBuffer.Length / 2)
{
Array.Copy(readBuffer, currentDecompPos, readBuffer, 0, currentReadPos -
currentDecompPos);
currentReadPos -= currentDecompPos;
currentDecompPos = 0;
}
if (length + currentReadPos - currentDecompPos > readBuffer.Length)
 {
byte[] oldBuf = readBuffer;
readBuffer = new byte[length + currentReadPos - currentDecompPos];
Array.Copy(oldBuf, currentDecompPos, readBuffer, 0, currentReadPos -
currentDecompPos);
currentReadPos -= currentDecompPos;
currentDecompPos = 0;
}
}
return 0;
}
}
```

In Java:

Here is sample program's loop:

```
public short ReceiveMsg (ArrayList<ExternalMsg> list) throws MsgException
{
      int length = 0;
      int output;
      try
      {
          readBuffer.clear();
          output = s.read (readBuffer);
          readBuffer.flip();
      }
      catch (Exception e)
      {
          output = -1;
      }
                  if (output <= 0)
                        return -1;
                  totalBytes += output;
                  readBuffer.get(myReadBuffer.buffer, myReadBuffer.getLimit(), output);
```

```
                    myReadBuffer.setLimit(myReadBuffer.getLimit()+output);
      while (myReadBuffer.remaining() > 5)
      {
         length = MSGGENFactory.getLen(myReadBuffer.buffer, myReadBuffer.position());
         int finalPos = length + myReadBuffer.position();
         if (length == 0)
            throw new MsgException(MsgException.MSG_ERROR_SOCKET_STX, "NO
Length");
         if (myReadBuffer.remaining() < length)
            break;
         ExternalMsg msg = MSGGENFactory.decompose(myReadBuffer);
        if (msg != null)
            list.add(msg);
        myReadBuffer.position(finalPos);
      }
        if (myReadBuffer.remaining() == 0)
      {
          myReadBuffer.clear();
          myReadBuffer.setLimit(0);
      }
      return 0;
          }
```

Once the application receives the list of ExternalMsg, each corresponding message is processed (by calling ExternalMsg's member function getMessageType()).

Message Processing

Market Data NBBO and Book Data

After login, the client is ready to send in Subscription for NBBO, BookData. Since the intention is to send out orders, the client subscribes for OrderUpdate and TradeUpdate for order status.

To subscribe for NBBO and BookUpdate:

In C++:

```
msggen::GlobalMsgSubscribe subMsg;
                    subMsg.init();
                    msggen::SubscriptionData *subData = new msggen::SubscriptionData();
                    subData->init();
                    subData->msgType = msggen::GlobalMsgTypeNBBOUpdate;
                    subData->security = symbolList[i];
                    subMsg.data.push_back(subData);
```

```
                subData = new msggen::SubscriptionData();
                subData->init();
                subData->msgType = msggen::GlobalMsgTypeBookUpdate;
        msggen::BaseMsg::setFlag (subData->bookflags,
msggen::GlobalBookDestinationALL);
                subData->security = symbolList[i];
                subMsg.data.push_back(subData);
```

In C#:

```
GlobalMsgSubscribe subMsg = new GlobalMsgSubscribe();
                subMsg.init();
                SubscriptionData subData = new SubscriptionData();
                subData.init();
                subData.msgType = GlobalMsgType.NBBOUpdate;
                subData.security = (string)symbolList[i];
                subMsg.data.Add(subData);
                subData = new SubscriptionData();
                subData.init();
                subData.msgType = GlobalMsgType.BookUpdate;
                BaseMsg.setFlag (ref subData.bookflags, (int)GlobalBookDestination.ALL);
                subData.security = (string)symbolList[i];
                subMsg.data.Add(subData);
```

In Java:

```
GlobalMsgSubscribe subMsg = new GlobalMsgSubscribe();
                subMsg.init();
        SubscriptionData subData = new SubscriptionData();
                subData.init();
                subData.msgType = GlobalMsgType.NBBOUpdate;
                subData.security = symbolList.get(i);
                subMsg.data.add(subData);
                subData = new SubscriptionData();
                subData.init();
                subData.msgType = GlobalMsgType.BookUpdate;
        subData.bookflags = BaseMsg.setFlag (subData.bookflags,
(int)GlobalBookDestination.ALL);
                subData.security = symbolList.get(i);
                subMsg.data.add(subData);
```

For NBBO subscription, the client will get continuous messages using GlobalMsgNBBOUpdate, for book subscription, the client will get continuous messages using GlobalMsgBookUpdate.

Order and Trade Updates

To subscribe for Order/Trade updates:

In C++:

```
msggen::GlobalMsgSubscribe subMsg;
                subMsg.init();
                msggen::SubscriptionData *subData = new msggen::SubscriptionData();
                subData->init();
                subData->msgType = msggen::GlobalMsgTypeOrderUpdate;
                subData->security = msggen::Const_msggen::GlobalWildCard;
                subData->firm = logonResp->data.userData.firm;
                subData->trader = logonResp->data.userData.trader;
                subData->fromTime = 1; //all orders
                subMsg.data.push_back(subData);
                subData = new msggen::SubscriptionData();
                subData->init();
                subData->msgType = msggen::GlobalMsgTypeTradeUpdate;
                subData->security = msggen::Const_msggen::GlobalWildCard;
                subData->firm = logonResp->data.userData.firm;
                subData->trader = logonResp->data.userData.trader;
                subData->fromTime = 1; //all trades
                subMsg.data.push_back(subData);
```

In C#:

```
GlobalMsgSubscribe subMsg = new GlobalMsgSubscribe();
                subMsg.init();
                SubscriptionData subData = new SubscriptionData();
                subData.init();
                subData.msgType = GlobalMsgType.OrderUpdate;
                subData.security = Const_msggen.GlobalWildCard;
                subData.firm = logonResp.data.userData.firm;
                subData.trader = logonResp.data.userData.trader;
                subData.fromTime = 1; //all orders
                subMsg.data.Add(subData);
                subData = new SubscriptionData();
                subData.init();
```

```
subData.msgType = GlobalMsgType.TradeUpdate;
subData.security = Const_msggen.GlobalWildCard;
subData.firm = logonResp.data.userData.firm;
subData.trader = logonResp.data.userData.trader;
subData.fromTime = 1; //all trades
subMsg.data.Add(subData);
```

In Java:

```
GlobalMsgSubscribe subMsg = new GlobalMsgSubscribe();
subMsg.init();
SubscriptionData subData = new SubscriptionData();
subData.init();
subData.msgType = GlobalMsgType.OrderUpdate;
subData.security = Const_msggen.GlobalWildCard;
subData.firm = logonResp.data.userData.firm;
subData.trader = logonResp.data.userData.trader;
subData.fromTime = 1; //all orders
subMsg.data.add(subData);
subData = new SubscriptionData();
subData.init();
subData.msgType = GlobalMsgType.TradeUpdate;
subData.security = Const_msggen.GlobalWildCard;
subData.firm = logonResp.data.userData.firm;
subData.trader = logonResp.data.userData.trader;
subData.fromTime = 1; //all trades
subMsg.data.add(subData);
```

Please note that the field trader comes from the logon response message. The above subscription gives all parent/child orders updates. Parent orders represent the order view from the client side perspective. Every time the client sends in a valid order entry, FIN will create one parent order. The child order represents the order view from the perspective of FIN's internal activities. For certain order types such as the smart router and algo order, FIN may create many child orders corresponding to the parent order. If the client wishes to receive parent orders only, the SubscriptionData's member field rootOnly option can be set to 'Y'.

The client will receive GlobalMsgOrderUpdate for OrderSubscription and GlobalMsgTradeUpdate for TradeSubscription

Order Actions

Next, to send out an order:

In C++:

```
msggen::GlobalMsgOrderAction *actionMsg = new msggen::GlobalMsgOrderAction;
        actionMsg->init();
        actionMsg->header.dest = msggen::GlobalDestinationSTAGE;
        actionMsg->data.actionType = msggen::GlobalOrderActionEntry;
        actionMsg->data.side = bors;
        actionMsg->data.quantity = qty;
        actionMsg->data.security = symbol;
        actionMsg->data.price = price;
        if (price == 0)
                actionMsg->data.orderType = msggen::GlobalOrderTypeMarket;
        actionMsg->data.category = dest;
        actionMsg->data.tif = tif;
```

In C#:

```
GlobalMsgOrderAction actionMsg = new GlobalMsgOrderAction();
        actionMsg.init();
        actionMsg.header.dest = GlobalDestination.STAGE;
        actionMsg.data.actionType = GlobalOrderAction.Entry;
        actionMsg.data.side = bors;
        actionMsg.data.quantity = qty;
        actionMsg.data.security = symbol;
        actionMsg.data.price = price;
        if (price == 0)
                actionMsg.data.orderType = GlobalOrderType.Market;
        actionMsg.data.category = dest;
        actionMsg.data.tif = tif;
```

In Java:

```
GlobalMsgOrderAction actionMsg = new GlobalMsgOrderAction();
        actionMsg.init();
        actionMsg.header.dest = GlobalDestination.STAGE;
        actionMsg.data.actionType = GlobalOrderAction.Entry;
        actionMsg.data.side = bors;
        actionMsg.data.quantity = qty;
```

```
actionMsg.data.security = symbol;
actionMsg.data.price = price;
if (price == 0)
actionMsg.data.orderType = GlobalOrderType.Market;
actionMsg.data.category = dest;
actionMsg.data.tif = tif;
```

Once the FIN server receives the order entry, the client receives order updates from the subscription regarding the order. For any trade that occurs, the client receives corresponding trade updates.

Please compile the sample program with corresponding library, and connect to FIN server for quote/order information.

Fraction Base Field for Order Actions

There is a fractionBase field

For example, if you want to send 0.01 share of BTCUSD,

Set quantity = 1, fractionbase=100 or quantity=10, fractionbase =1000, which both are 0.01 share.

There is no normalized fraction base set in the system, so set BTCUSD to 100000, meaning the minimum allowable qty for BTCUSD is 1/100,000 share.

When you send above 0.01 share into system, on return, your order will be normalized, you will get quantity=1000 and fractionbase=100000 instead of what you input quantity=1 and fractionbase=100.

For Example:

Send to Neutron:

```
"{\"security\":\"Microsoft\",\"tif\":1,\"side\":\"S\",\"category\":0,\"price\":60,\"actionType\":0,\"execQty\":100,\"fractionBase\":1000,\"clientRef1\":\"TRADER.115\"}"
```

For Bitcoin, we use to set the value at 10,000. In the admin, we should have something for the symbols field that allows FractionBase to be set for a symbol when a symbol is a cryptosecurity like Bitcoin or a tokenized asset.

Example of an Order

firm=ATLA|security=BTC 20151231C 2000.000|category=STAGE|orderType=Market|actionType=Entry|side=S|price=0.0|tif=GTC|clientRef1=844-081414-192942-028|clientRef2=WEB|userid=ATLASATS1|trader=844|quantity=2000|instrumentType=Options|fractionBase=10000|

FIN Native Messages

FIN native API messages are backward and forward compatible. New messages and new fields can be added to the API but existing messages and fields will never change so users of previous API versions are not be affected by new changes. If a user wants to access to new messages or fields, they can simply download a new library.

The following are some commonly used messages. This is intended to be a brief introductory document, therefore (even for those commonly used messages) only some basic fields and functionality will be listed here.

```
BaseMsg: provides utility functions for field validation
boolean isFieldValid()
```

This class contains utility functions such as BaseMsg.isFieldValid, which can be used to check whether a field contains any valid value before using it

```
ExternalMsg: base class for all messages, abstract
getMessageType (); //returns the message type (GlobalMsgType) of the underlying message,
ExternalTranMsg: base class for all transaction messages, abstract
```

All ExternalTranMsg has a field header of GlobalMsgTranHeader, containing the following fields:

```
public int tranId;
public GlobalResult tranResponse;
public String errorDetails;
```

The client should always receive a response to the request. If the transaction request is successful, tranResponse will be GlobalResult.SUCCESS, otherwise, tranResponse will be GlobalResult.FAILURE and errorDetails is likely to contain some level of explanation for the failure. This field should always be checked first before processing transaction response.

If tranId is set to a certain value, the response message will also carry that value. It helps to match the original request.

```
ExternalDataMsg:  is a base class for all transaction messages, abstract
```

All ExternalDataMsg have a field header of GlobalMsgDataHeader, containing the following fields:

```
public int subId;
```

subId carries the subId value in SubscriptionData when the corresponding subscription is set to message. This will help to match up the original subscription.

Logon

```
GlobalMsgLogon extends ExternalTranMsg
```

Logon msg is the first msg required to send to the server once a connection is established. At minimum, the firm and user id in the logon request (see example) should be provided.

Additionally a few parameters can be set in Logon data

- HeartbeatInterval, in seconds, set this field if you want server to send you heartbeat.
- AggregateTime, in milliseconds, set this field if you want server to send you an update NBBO and book feed for that time period instead of each tick update. Maximum is 10 seconds.
- ThresholdLevel, num of Level 2 data that you want to receive, only apply to BookUpdate subscription.

```
GlobalMsgSubscribe extends ExternalTranMsg
GlobalMsgUnsubscribe extends ExternalTranMsg
```

Market Data

Both GlobalMsgSubscribe and GlobalMsgUnsubscribe contain a list of SubscriptionData objects. Each SubscriptionData contain a subscription (or unsubscription) request.

To subscribe, use the GlobalMsgSubscribe.

To unsubscribe, use GlobalMsgUnsubscribe and pass the same parameters that are set in GlobalMsgSubscribe.

GlobalMsgNBBOUpdate extends ExternalDataMsg

It contains NBBOData information; here are some of NBBOData fields

Field	Req'd	Type
Security	Y	String
BidPrice	Y	Double
BidSize	Y	Int
AskPrice	Y	Double
AskSize	Y	Int
Status	Y	GlobalQuoteStatus
SeqNum	Y	Int
ExchangeTime	Y	Int
RegSHOInd	Y	Char

Field status indicates the status of Quote. If everything is normal, it will have Normal status. If quote is closed, it will have closed status.

To subscribe, set SubscriptionData.msgType = GlobalMsgType.NBBOUpdate, and security field to corresponding symbol.

GlobalMsgBookUpdate extends ExternalDataMsg

It contains a list of BBOData, here are some of BBOData fields

BBODATA

Field	Req'd	Type
Side	Y	Char
Security	Y	String
Booksource	Y	Globalbookdestination
MarketMaker	Y	String
Price	Y	Double
Status	Y	Globalbookstatus
Keystr	Y	String
ID	Y	Long
Bookqty	Y	Int

Seqnum	Y	Long
ExchangeTime	Y	Int

Field Status - Field status indicates the meaning of the message.

Add - indicates that a new entry has been created. Update indicates an update to an existing entry. Remove indicates that the entry no longer exists.

Complete indicates the completion of the initial snapshot transmission. The data itself does not carry any book entry.

RemoveAll indicates failure of certain book feed, client should remove book entries for that corresponding feed (as indicated in bookSource field). Also, both keyStr and id field can be used as key to search for previous entry. Both are guaranteed to be unique within symbol subscription.

To subscribe - set SubscriptionData.msgType = GlobalMsgType.BookUpdate, and security field to corresponding symbol.

You can subscribe for all servers' available book feeds by setting ALL flag in SubscriptionData.bookFlag field,

```
BaseMsg.setFlag (ref subData.bookflags, (int)GlobalBookDestination.ALL);
```

Or you can specify individual feeds

```
BaseMsg.setFlag (ref subData.bookflags, (int)(GlobalBookDestination.BBO|
GlobalBookDestination.INET| GlobalBookDestination.NYSE));
GlobalMsgLastSaleUpdate extends ExternalDataMsg
GlobalMsgSimpleLastSaleUpdate extends ExternalDataMsg
```

Fraction Base for Book Data

Subscribe side is on BookData fraction qty logic.

A new field fractionBase is added to BookData structure.

Here is the new logic.

```
double entryQty;
If (BaseMsg.isFieldValid (bookData.bookQtyLong))
{
```

```
If (BaseMsg.isFieldValid (bookData.fractionBase))
      entryQty = bookData.bookQtyLong/bookData.fractionBase;
            else
      entryQty = bookData.bookQtyLong;
}
Else
{
       entryQty = bookData.bookQty;
}
```

Both GlobalMsgLastSaleUpdate and GlobalMsgSimpleLastSaleUpdate carry LastSale information. The difference is if the last sale is simple enough, such as there is no new high/low/open/closing, GlobalMsgSimpleLastSaleUpdate is used. Otherwise, GlobalMsgLastSaleUpdate will be used.

Indictor and additional Indicators are sale conditions for the trades. They are corresponding SIP/SIAC value. lastExchange field indicate which exchange the trade happened.

Here is exchange code

```
        switch (v)
        {
        case 'A':
            return AMEX_String;
        case 'B':
            return BOSX_String;
        case 'W':
            return CBOE_String;
        case 'C':
            return CINN_String;
        case 'D':
            return FNRA_String;
        case 'M':
            return MWSE_String;
        case 'N':
            return NYSE_String;
        case 'X':
            return PHLX_String;
        case 'P':
            return ARCA_String;
```

```
        case 'I':
            return ISEX_String;
        case 'T':
            return NASD_String;
        case 'Z':
            return BATS_String;
        case 'Y':
            return BATY_String;
        case 'J':
            return EDGA_String;
        case 'K':
            return EDGX_String;
        }
```

GlobalMsgLastSale status update

If there is cancel/correction, it will be in GlobalMsgLastSale with the following status: Normal, Cancel, Restate, Prior Trade, Prior Cancel, and Prior Restate.

Definitions:

- PriorTrade/PriorCancel/PriorRestate is either a missed reported
- Trade/cancel/correct that happens on a prior day (priorDate and priorTime fields indicate when refereed trade happens)
- Cancel/Restate is on a cancel/correct on today's trade (priorTime field indicate when referred trade happens)

SubscriptionCompleteUpdate message

When a client subscribes to the wildcard, and it is used with OrderUpdate and TradeUpdate subscription, it is used to inform you that you have received all of the cache data.

To subscribe, set SubscriptionData.msgType = GlobalMsgType.LastSaleUpdate, and security field to corresponding symbol. First message that arrives should be GlobalMsgLastSaleUpdate as it carries as much info as possible, then depending on data contents, you may receive either SimpleLastSaleUpdate or LastSaleUpdate.

Symbol Query

GlobalMsgDBSecmstQuery to query symbols.

LastSaleData

Field	Req'd	Type
Security	Y	String
Price	Y	double
Quantity	Y	Int
Status	Y	Enum, Open,Cancel,Correct
TradeThroughExempt	N	Char
SaleDays	N	GlobalQuoteStatus
SeqNum	Y	Int
ExchangeTime	Y	Int
TotalVolume	Y	Long (64 bits)
HighPrice	N	Double
LowPrice	N	Double
OpenPrice	N	Double
PrevClosePrice	Y	Double
ClosingPrice	N	Double,
ExchangeTime	Y	Int
LastExchange	Y	Char

SimpleLastSaleData

Field	Req'd	Type
Security	Y	String
Price	Y	Double
Quantity	Y	Int
Status	Y	Enum, Open,Cancel,Correct
TradeThroughExempt	N	Char
SaleDays	N	GlobalQuoteStatus
SeqNum	Y	Int
ExchangeTime	Y	Int
TotalVolume	Y	Long (64 bits)
ExchangeTime	Y	Int
LastExchange	Y	Char

History Market data

GlobalMsgLastSaleMinuteQuery extends ExternalTranMsg

Use this message to query for last sale minute summary for the security. You must provide security field in data sub structure. If you want to query previous day's data set date field in format of "MM/DD/YY". You can use beginTime/endTime field to set the filter for your result.

- BeginTime/endTime specifies milliseconds since midnight.
  - For example, if you only want data after 9:30AM, set beginTime to 93000000.
  - If successful, result carries a list of LastSaleMinuteSummaryData.

For each record, it carries info such as high/low/number of trades/total share in that minute. Field exchangeStartTime indicates the milliseconds since. Its format is HHMMSS000, indicating number milliseconds since midnight. If the data is for minute of 12:15, exchangeStartTime will have value 121500000.

GlobalMsgNBBOMinuteQuery extends ExternalTranMsg

Use this message to query for NBBO minute summary for the security. You must provide security field in data sub structure. If you want to query previous day's data set date field in format of "MM/DD/YY". You can use beginTime/endTime field to set the filter for your result.

- BeginTime/endTime specifies milliseconds since midnight.
  - For example, if you only want data after 9:30AM, set beginTime to 93000000.
  - If successful, result carries a list of LastSaleMinuteSummaryData.

For each record, it carries info such as high/low price and size of bid and ask in that minute. Field exchangeStartTime indicate the millisecond since midnight. Its format is HHMMSS000, indicating number milliseconds since midnight.

If the data is for minute of 12:15, exchangeStartTime will have value 121500000.

GlobalMsgLastNBBOSecondQuery extends ExternalTranMsg

User this message to query for NBBO/Lastsale second snapshot

You must provide security and date field in data sub structure. The date field is an integer field; its expected format is YYMMDD.

For example, if you want to search for info in Nov 3rd, 2010, date field should be set to 101103.

You can use beginTime/endTime field to set the filter for your result.

- BeginTime/endTime specifies milliseconds since midnight.

- For example, if you only want data after 9:30:01AM, set beginTime to 93000100.

You can use data.filter field to selectively retrieve the second (0-59, inclusive) data that you interested, such as “0,5,10,30” means you want to retrieve 0th,5th, 10th and 30th second data only.

If successful, result carries a list of LastNBBOSecondData.

For each record, it carries info such as bid/ask/lastPrice/cumQty of that second. cumQty is the total trade qty since opening. Field exchangeStartTime indicate the millisecond since midnight. Its format is HHMMSS000, indicating number milliseconds since midnight. If the data is for minute of 12:15, exchangeStartTime will have value 121500000.

Request example of a Last Minute Query

```
msgType=LastSaleMinuteQuery|tranId=10|lifeTime=30|security=AA|data=0|
```

Response example of a Last Minute Query

```
msgType=LastSaleMinuteQuery|firm=ROOT|dest=NYSEALS|tranId=10|globalID=2276|lifeTime=30|flags=4|type=DefClient|tranResponse=SUCCESS|security=AA|data=350|data[0]=[security=AA|exchangeStartTime=70000000|highPrice=9.75|lowPrice=9.75|volume=1200|numOfTrades=3|lastPrice=9.75|amount=11700.075||]data[1]=[security=AA|exchangeStartTime=70500000|highPrice=9.78|lowPrice=9.78|volume=100|numOfTrades=1|lastPrice=9.78|amount=978.0|]data[2]=[security=AA|exchan0|geStartTime=70900000|highPrice=9.79|lowPrice=9.79|volume=200|numOfTrades=1|lastPrice=9.79|amount=1958.0|]data[3]=[security=AA|exchangeStartTime=71000000|highPri9|ce=9.79|lowPrice=9.79|volume=1000|numOfTrades=1|lastPrice=9.79|amount=9790.0|]data[4]=[security=AA|exchangeStartTime=71200000|highPrice=9.75|lowPrice=9.75|volume=400|numOfTrades=1|lastPrice=9.75|amount=3900.0|]data[5]=[security=AA|exchangeStartTime=74300000|highPrice=9.76|lowPrice=9.76|volume=100|numOfTrades=1|lastPrice=9.76|amount=976.0|]data[6]=[security=AA|exchangeStartTime=74800000|highPrice=9.76|lowPrice=9.76|volume=3500|numOfTrades=1|lastPrice=9.76|amount=34160.0|]data[7]=[security=AA|exchangeStartTime=74900000|highPrice=9.76|lowPrice=9.76|volume=16400|numOfTrades=14|lastPrice=9.76|amount=160064.0|]data[8]=[security=AA|exchangeStartTime=75400000|highPrice=9.78|lowPrice=9.78|volume=500|numOfTrades=2|lastPrice=9.78|amount=4890.0|]data[9]=[security=AA|exchangeStartTime=75600000|highPrice=9.77|l:
```

Order Actions

```
GlobalMsgOrderAction extends ExternalTranMsg
```

ActionType, required, enum, GlobalOrderAction.Entry for new order entry request, GlobalOrderAction.Cancel for order cancel, GlobalOrderAction.Replace for order correction.

- Side, required, char, 'B' for buy order, 'S' for sell order.
- ShortSaleFlag, optional, enum, GlobalShortSaleFlag.ShortSale for short sale, GlobalShortSaleFlag.ShortSaleExempt for short sale exempt. If it is a shortsale order, you must specify the corresponding flag
- Security, required, string, symbol for the instrument
- InstrumentType, optional, enum, default is GlobalInstrumentType.Equity,
    - For options order, set it to enum GlobalInstrumentType.Option
    - For Futures order, set it to enum GlobalinstrumentType.Future
    - For FX order, set it to enum GlobalinstrumentType.FX
    - For crypto order, set enum GlobalInstrumentType.Crypto
- Quantity, required,int, order quantity
- Price, required, double, for entry/replace, price of order. If it is market order, set to 0.
- Instruction, optional, string, this is to specify how you want to order routed out.

Marking Order Based upon Position

If no position enforcement is to be used then API to set the following flag when sending out the order, and it will mark with shortsale flag based on the position.

```
Msg.data.orderFlags2 = BaseMsg.setFlag(msg.data.orderFlags2,
Const_msggen.GlobalOrderFlag2MarkShort);
```

Instrument Type example:

There is a field instrumentType in OrderData, which need to be set for corresponding instrument type. If not set, default assumes it is Equities.

```
enum GlobalInstrumentType
{
    Equities,
    Options,
    Futures,
    FX,
   Crypto,
};
```

For directed order use "SSD:" instruction set. For example, send order to INET, set instruction field to "SSD:INET", send order to ARCA, set instruction field to "SSD:ARCA", etc…

If no instruction is specified, the order will be staged within FIN, and will be routed out through FIN's smart router when there is eligible quote.

- OrderType, optional, enum
  - If it is market order, set to GlobalOrderType.Market
- Tif, optional, enum
  - Default is GlobalTimeInForce.Day
  - Additional value: IOC, Gtx, Gtt. If value is Gtt, set tifValue field the corresponding expiration time. ExpireTime is express in milliseconds since UTC Jan 1st, 0:0:0, 1970 (similar to UNIX's UTC time, except in milliseconds)
- ClientRef1, optional, string
  - You can set a reference number for you to track your own order. It will be sent back to you in OrderUpdate/TradeUpdate

Use this message to send order entry/cancel/replace to server.

GlobalOrderUpdate extends ExternalDataMsg
OrderUpdate contains a list of OrderData

OrderData carries information that user sends in, such as security, side, quantity, trader, userid, etc.

- Category, enum, order category, such as INET/ARCA/NYSE, etc…
- Refno, string, the order reference number, uniquely assigned by FIN backend.
- RootRef, string, the order reference number of root order. For parent order, rootRef is same as refno.
- OrderStatus, enum, status of order, such as Pending, Open, PendingCancel, Canceled, Rejected, Executed, etc…
- LiveQty, int, the quantity that is still live.
- ExecQty, int, the quantity that is executed so far.
- RejectQty,int, that quantity that is not executed when order is closed.
- ClientRef1, string, if customer set the clientRef1 field when sending new order, the corresponding value is passed back here.

OrderData

Field	Req'd	Type
Firm	Y	String
Security	Y	String
Category	Y	Enum, INET/ARCA/BATS/STAGE,etc…
OrderStatus	N	Enum, Open/Pending/PendingCancel,etc…
Price	Y	Double

Quantity	Y	Int
Side	Y	Char
StopPrice	N	Double
PegType	N	Enum, Primary/Reverse/Mid,etc…
OrderCapacity	N	Enum, Principal/Agency,etc…
Tif	N	Enum, DAY/GTX/IOC,etc…
Refno	Y	String
LiveQty	Y	Int
RejectQty	Y	Int
ExecQty	Y	Int
ExecQty	Y	Int
DisplayQty	Y	Int
ExecDollarAmount	Y	Double
Text	N	String
ClientRef1	N	String
Trader	Y	String

Sending an Order Example

2017-03-03 15:42:55,888 - New order for 'XOM' has been placed

Security: XOM, Side: B, Price: 300, Quantity: 200, actionType: Entry, category: TEST, tif: DAY

ORDER and TRADE Update Subscription messages

Order Data(2017-03-03 15:42:58,419) Security: XOM, RefNo: 4UK6YR48O4Y9C7, Price: 300, CalcPrice: 1.79769313486232E+308, Quantity: 200, QuoteQty: 2147483647, QuotePrice: 1.79769313486232E+308, Commission: 1.79769313486232E+308, Side B, ExecQty: 2147483647, execDollarAmount: 1.79769313486232E+308, buyingEffectQty: 2147483647, carrExecQty: 2147483647, displayQty: 2147483647, liveQty: 2147483647, minQty: 2147483647, origQty: 200, rejectQty: 2147483647, restatedQty: 2147483647, stageLiveQty: 2147483647, seqNum: 2147483647, Trader: TEST1

Order Data(2017-03-03 15:42:58,420) Security: XOM, RefNo: 4UK6YR48O4Y9C7, Price: 300, CalcPrice: 1.79769313486232E+308, Quantity: 200, QuoteQty: 2147483647, QuotePrice: 1.79769313486232E+308, Commission: 1.79769313486232E+308, Side B, ExecQty: 0, execDollarAmount: 0, buyingEffectQty: 2147483647, carrExecQty: 2147483647, displayQty: 2147483647, liveQty: 200, minQty: 2147483647, origQty: 200, rejectQty: 2147483647, restatedQty: 2147483647, stageLiveQty: 2147483647, seqNum: 1594, Trader: TEST1

Trade Data(2017-03-03 15:42:58,420) Security: XOM, RefNo: 4UK6YR48O4Y9C7_21474836474175_E, ExecPrice: 300, ExecQty: 200, PrevPrice: 1.79769313486232E+308, PrevQty: 2147483647, Side: B, Commission 1.79769313486232E+308, seqNum: Open, executionType: 1595, orderAdjusted: 2147483647, orderRef:4UK6YR48O4Y9C7 , userid: TEST1

Order Data(2017-03-03 15:42:58,421) Security: XOM, RefNo: 4UK6YR48O4Y9C7, Price: 300, CalcPrice: 1.79769313486232E+308, Quantity: 200, QuoteQty: 2147483647, QuotePrice: 1.79769313486232E+308, Commission: 1.79769313486232E+308, Side B, ExecQty: 200, execDollarAmount: 0, buyingEffectQty: 2147483647, carrExecQty: 2147483647, displayQty: 2147483647, liveQty: 0, minQty: 2147483647, origQty: 200, rejectQty: 2147483647, restatedQty: 2147483647, stageLiveQty: 2147483647, seqNum: 1595, Trader: TEST1

Setting Commission on Order/Trades

Commission

There is a default firm level fee. Fees are applied on the user account. Fee is deducted from the position in the receiving coin (example below)

Example 1Commission:

Client 1 has the default fee of 2%
Client 2 has a discounted fee of 1%

They do a trade with one another..
Client 1 buys 1 BTC with 20 ETH
Client 2 buys 20 ETH with 1 BTC

Result: fee / positions:
Client 1 Position Adjustments (net of fees): 20 ETH decrease, 0.98 BTC increase
Client 2 Position Adjustments (net of fees): 19.8 ETH increase, 1 BTC decrease
Net market gain after deductions:
(0.2 ETH, .02 BTC)

Example 2 Commission:

If the commission rate is set to 0.02 for the USER01.

The commission will return to you in ExecutionReport with 12(commission) as value of the commission taken, 479(commion currency) set to corresponding incoming currency, such as “BTC”, “USD”. For tag value 12, it is real value, not the fraction based value, as underlying currency can have different fraction base from the currency pair.

For example, a trade for 1 share of BTCUSD@9000, the buyer will pay commission of $180.00 in USD, seller will pay commission of 0.02 in BTC. Buy order will have 12 as 180.00 and 479 as “USD”. Sell order will have 12 as 0.02 and 479 as “BTC”.

Next step is to define how you want to add user/update user info. SettementInstructions is used.

The commision can be set per order or through user configuration.

If set through user configuration, in API, send a GlobalMsgDBFirmUserUpdate where you update corresponding action.data.commission field. (you can see corresponding value in pretrade risk screen’s commission column)

Or, you can set the commission per order. To do that, set orderAction.data.commission field.

It is a double value, say 0.004 means take 0.4% commission.

Also, if client want to specify that liquidity provider won’t need to pay commission, then for incoming order, should set flag

```
ordAction.data.orderFlags4 = BaseMsg.setFlag (ordAction.data.odrerFlags4,
Const_msggen.GlobalOrderFlag4NoFeeOnMaker);
```

Parent & Child Order Data

Comment out the rootOnly='Y', and you will get both parent and child orders.

Example:

```
GlobalMsgSubscribe subMsg = new GlobalMsgSubscribe();
 subMsg.init();
 final long nowTs = System.currentTimeMillis();
 String trader = logonMsResponse.data.userData.trader;

 SubscriptionData subData = new SubscriptionData();
 subData.init();
 subData.msgType = GlobalMsgType.OrderUpdate;
 subData.security = Const_msggen.GlobalWildCard;
 subData.firm = logonMsResponse.data.userData.firm;
 subData.rootOnly = 'Y'; // to cut down traffic.
 subData.trader = trader;
 subData.fromTime = nowTs; // use 1 for all historical orders
 subMsg.data.add(subData);
```

```
subData = new SubscriptionData();
subData.init();
subData.msgType = GlobalMsgType.TradeUpdate;
subData.security = Const_msggen.GlobalWildCard;
subData.firm = logonMsResponse.data.userData.firm;
subData.trader = trader;
subData.fromTime = nowTs; // use 1 for all historical orders
// subData.security =
subMsg.data.add(subData);
int v = socket.sendMsg(subMsg);
return v;
```

Trade Updates

Subscription to Trade is very similar to Order, except you request for GlobalMsgType.TradeUpdate instead.

- TradeStatus, enum, normally it is GlobalTradeStatus.
- Canceled for broken trade or GlobalTradeStatus.Replaced for corrected trade.
- In order to track the information the ClientRef1, string. If customer set the clientRef1 field when sending new order, the corresponding value is passed back here.

```
GlobalTradeUpdate extends ExternalDataMsg
```

- TradeUpdate contains a list of TradeData.
- Category, enum, order category, such as INET/ARCA/NYSE, etc…
- Refno, string, unique identifier assigned by FIN server
- OrderRef, string, corresponding order reference number, through this field, you can find corresponding order that trade is generated from.
- RootRef, string, corresponding parent order's reference number.
- ExecQty, int, the number of shares executed.
- ExecPrice, double, the price of execution.

TradeStatus, enum, normally it is GlobalTradeStatus.Normal, but it can be GlobalTradeStatus.Canceled for broken trade, or GlobalTradeStatus.Replaced for corrected trade .

ClientRef1, string. If customer set the clientRef1 field when sending new order, the corresponding value is passed back here.

Subscription to Trade is very similar to Order, except you request for GlobalMsgType.TradeUpdate instead.

TradeData

Field	Req'd	Type
Firm	Y	String
Security	Y	String
Category	Y	Enum, INET/ARCA/BATS,etc…
TradeStatus	N	Enum, Open/Cancel/Correction
ExecPrice	Y	Double
ExecQty	Y	Int
Side	Y	Char
OrderCapacity	N	Enum, Principal/Agency,etc…
Refno	Y	String
PrevRefno	N	String, link to previous trade if cancel or correct
OrderRef	Y	String
ClientRef1	N	String
Trader	Y	String

Cancel/Replace Orders

In order to replace an order, the following would be used actMsg.data.actionType = GlobalOrderAction.Replace. The following need to be referenced along with the action type, security,side , refno, rootRef field from existing order. If you need to change quantity/price, set field to new value, otherwise copy the existing info.

An example of how to set

The clientRef1 is for that purpose. In new order, you set clientRef1=A. In cancel replace, you should set clientRef1=B. Corresponding

ClientRef1 will come back to you in OrderUpdate and TradeUpdate if you subscribe to them. You don't set it in cancel request.

Examples of API Message Flow.

<== Sending Order Information Via API to Neutron

==> Neutron is sending back information There are two types of actions (a: action, u: update)

```
16:24:31,798 INFO  SEND order <==
firm=WTSD|security=S|category=STAGE|orderType=Limit|actionType=Entry|side=B|pri
```

ce=1|tif=GTX|clientRef1=1624000036|1|clientRef2=NITEFAN|clientRootId=1624000036|trader=WTS0006|clientTif=GTX|quantity=100|instruction=SSD:NITEEDGE|fixTags=21=1 57=ALGO 6101=FAN 6210=0
|account=24438|symbolFormat=ACT|comment=NITEFAN|

16:24:31,798 INFO RECV action ==> a:
firm=WTSD|security=S|category=STAGE|orderStatus=Open|orderType=Limit|actionDir=STAGE|actionType=Entry|side=B|price=1|orderCapacity=Agency|tif=GTX|complexOrigin=A5|baseType=Y|refno=3NW8RD07E91DA5|hierarchyRef=3NW8RD07E91D|rootRef=3NW8RD07E91DA5|clientRef1=1624000036|1|clientRef2=NITEFAN|clientRootId=1624000036|1|userid=WTSD|trader=WTS0006|clientTif=GTX|quantity=100|origQty=100|liveQty=100|execQty=0|execDollarAmount=0|stageLiveQty=0|instruction=SSD:NITEEDGE|seqNum=1|recno=0|orderFlags=6442450944|date=12/21/12|entered=1356125071798|updated=1356125071798|multiDayFlag=N|fixTags=21=1 57=ALGO 6101=FAN 6210=0
|account=24438|buyingPowerLimit=1|symbolFormat=ACT|chainID=3NW8RD07E91DA5|comment=NITEFAN|

16:24:31,798 INFO RECV update ==> u:
firm=WTSD|security=S|exchange=NYSE|category=STAGE|orderStatus=Open|orderType=Limit|actionDir=STAGE|actionType=Entry|side=B|price=1|orderCapacity=Agency|tif=GTT|tifValue=1356145202876|complexOrigin=A5|baseType=Y|refno=3NW8RD07E91DA5|hierarchyRef=3NW8RD07E91D|rootRef=3NW8RD07E91DA5|clientRef1=1624000036|1|clientRef2=NITEFAN|clientRootId=1624000036|1|userid=WTSD|trader=WTS0006|clientTif=GTX|quantity=100|origQty=100|liveQty=100|execQty=0|execDollarAmount=0|stageLiveQty=0|instruction=SSD:NITEEDGE|seqNum=1|recno=0|orderFlags=6442450944|date=12/21/12|entered=1356125071798|updated=1356125071798|multiDayFlag=N|fixTags=21=1 57=ALGO 6101=FAN 6210=0
|account=24438|nbboBid=5.46|nbboAsk=5.47|nbboSeqNum=20904248|nbboTime=1356125071798|buyingPowerLimit=1|symbolFormat=ACT|chainID=3NW8RD07E91DA5|comment=NITEFAN|nbboBidSize=36|nbboAskSize=1|

Send Cancel Request

16:24:40,034 INFO SEND cancel <==
firm=WTSD|security=S|category=STAGE|orderStatus=Pending|orderType=Limit|actionType=Cancel|side=B|price=1|stopPrice=0|tif=GTX|refno=3NW8RD07E91DA5|rootRef=3NW8RD07E91DA5|clientRef2=NITEFAN|trader=WTS0006|quantity=100|instruction=SSD:NITEEDGE|entered=1356125071798|updated=1356125080013|account=24438|symbolFormat=ACT|comment=NITEFAN|

16:24:40,034 INFO RECV action ==> a:
firm=WTSD|security=S|category=STAGE|orderStatus=Pending|orderType=Limit|action Type=Cancel|side=B|price=1|stopPrice=0|tif=GTX|refno=3NW8RD07E91DA5|rootRef= 3NW8RD07E91DA5|clientRef2=NITEFAN|trader=WTS0006|quantity=100|instruction=S SD:NITEEDGE|entered=1356125071798|updated=1356125080013|account=24438|sy mbolFormat=ACT|comment=NITEFAN|

16:24:40,034 INFO RECV update ==> u:
firm=WTSD|security=S|exchange=NYSE|category=STAGE|orderStatus=Canceled|orderT ype=Limit|actionDir=STAGE|actionType=Entry|side=B|price=1|orderCapacity=Agency|t if=GTT|tifValue=1356145202876|complexOrigin=A5|baseType=Y|refno=3NW8RD07E91 DA5|hierarchyRef=3NW8RD07E91D|rootRef=3NW8RD07E91DA5|exchangeRef=085910 294|clientRef1=1624000036|1|clientRef2=NITEFAN|clientRootId=1624000036|1|oatsC ancelTime=1356125080034|userid=WTSD|trader=WTS0006|clientTif=GTX|quantity=10 0|origQty=100|liveQty=0|execQty=0|execDollarAmount=0|rejectQty=100|stageLiveQt y=0|text=CANCELED|instruction=SSD:NITEEDGE|seqNum=3|recno=0|orderFlags=23622 320128|date=12/21/12|entered=1356125071798|updated=1356125080034|multiDayF lag=N|fixTags=21=1 57=ALGO 6101=FAN 6210=0
|account=24438|nbboBid=5.46|nbboAsk=5.47|nbboSeqNum=20904248|lastInTags=1= WTS0006|nbboTime=1356125071798|buyingPowerLimit=1|symbolFormat=ACT|chainI D=3NW8RD07E91DA5|comment=NITEFAN|nbboBidSize=36|nbboAskSize=1|

Order Types

There are three separate places where one specifies where an order is going:

- actionMsg.header.dest
- actionMsg.data.category
- actionMsg.data.instruction

The header.dest specifies where this message goes. For OrderAction, it normally is STAGE. Data.category and data.instruction combined define what this order is. There are multiple ways to do the same task, so it depends on what a client wants to do.

For simple pass through, a client can specify data.cateogry=GlobalDestination.NYSE. The system will generate one order and that order will go to NYSE. In this case, FI will only act as pre-risk check and routing purpose.

For a more complicated order, such as smart order/synthetic peg order and etc… There is possibility FI may need to generate more than 1 order for customer order, in that case a client will specify data.category = GlobalDestination.STAGE and instruction, orderType and etc…

The following is an example of sending a NYSE limit-on-open:

- Set the normal price, qty, symbol fields, set actMsg.data.category = GlobalDestination.NYSE; actMsg.data.openCloseType = GlobalOpenCloseType.OnOpen;
- If the client needs to replace the order,actMsg.data.actionType = GlobalOrderAction.Replace;
- Client also needs to copy security, side, refno, rootRef field from existing order.
- If client needs to change quantity/price, set field to new value, otherwise copy the existing information.

Notional Value Orders

A new flag (Const_msggen.GlobalOrderFlag4UseActualAmount) is added to msggn.jar library, please download the latest library.

To send by amount order

```
actMsg.data.orderType = GlobalOrderType.Market;

actMsg.data.orderFlags4 =
BaseMsg.setFlag(actMsg.data.orderFlags4,Const_msggen.GlobalOrderFlag4UseActualAmount);

actMsg.data.actualExecQty = 5000.10; //that is 5000.10 currency amount in corresponding crypto pair currency amount. If BTCUSD, that 5000.1 USD. If BTCEUR, that is 5000.1 EUR.
```

Because it is possible that order may execute in multiple price bands, so system will update quantity,liveQty fields correspondingly to reflect that.

Post Only Order

Set the following flag in order for post-only order to work

```
BaseMsg.setFlag(data.orderFlags4, Const_msggen.GlobalOrderFlag4OrderCRYPTO);
```

Routing Instructions

The following is how to set up the Routing Instructions data tab on the console for the preference list for the smart router.

- Stage, Firm, Security, Subject, Destination, Alt Destination,
- ROUT, *, *. INET, ARCA
- ROUT, *, *, ARCA, ARCA INET
- ROUT, *,*, BATS, BATS, INET
- ROUT, *,*, EDGX, EDGX, INET
- ROUT, *, *, NYSE, NYSE, INET

Otherwise the API can be used for this as well the message GlobalMsgDBRoutingInstUpdate to change the route.

The following is an example of routing instructions update:

```
msgType=DBRoutingInstUpdate|firm=LEHM|dest=DB|tranId=4|flags=4|stage=ROUT|firm=*|security=*|subject=NYSE|destination=ARCA|alterDestination=NYSE|
```

Example of smart router in action:

By default, orders staged within FIN will be automatically sent to smart router if order price is better than NBBO. In this example, we send buy 20000 DELL MKT to FIN, the root order will automatically send it to smart router (ROUT), which will then send out orders to different venues based on protected quotes. Smarter router will work the book until the order itself is fully executed or market is out of reach.

Here is a configuration snapshot:

- “P” is for parallel, “S” is for serial, “A” is for automatic
- Console settings in Routing Instruction Tab
- Stage, Firm, Security, Subject, Destination, Routing Instruction
- Stage, DEMO, GOOG, P, Dark 50, Dark2, 50 (note, pinging two routes by splitting order)

More complex routing instructions can be created this would require to set the OrderAction.orderData’s field routingSessionName to “Custom Name” when sending order.

The following is examples of how to generate ALGO orders:

OCO example:

- OrderData.executionType = GlobalExecutionType.OCO;
- OrderData..manning_refno = uniqueID //unique id of the group

VWAP example:

- OrderData.executionType = GlobalExecutionType.BasicVWAP;
- OrderData.numOfSlices = v;//num of slices you want to take
- OrderData.randomSeed = v1;//random number so system can randomize to hide your from being detected

TWAP example:

- OrderData.executionType = GlobalExecutionType.BasicTWAP;
- OrderData.numOfSlices = v;//num of slices you want to take
- OrderData.sliceThreshold = vshares;//share of last sale for system to launch next slice
- OrderData.randomSeed = v1;//random number so system can randomize to hide your from being detected

Multileg Orders

Use GlobalMsgMultiLegOrderAction for multi-leg orders

Example of Order Entry for Multileg

```
msgType=MultiLegOrderAction|firm=ABNA|dest=STAGE|tra
nId=2|flags=4|actionType=Entry|quantity=100|orderType=Limit|tif=DAY|category=INE
T|firm=ABNA|price=1.01|data=2|data[0]=[firm=ABNA|security=JNJ 20120121P
50.000|category=STAGE|actionType=Entry|side=B|price=1.01|tif=DAY|trader=MULTI|
quantity=100|instrumentType=Options|]data[1]=[firm=ABNA|security=JNJ 20120121C
60.000|category=STAGE|actionType=Entry|side=S|price=1.
01|tif=DAY|trader=MULTI|quantity=100| instrumentType=Options]
```

Example of Order Cancel for Multileg

The cancel is still just OrderAction message

```
23:03:39.439-(TRAN[I->19]):
msgType=OrderAction|firm=ABNA|dest=STAGE|tranId=8|fl
ags=4|firm=ABNA|security=JNJ|actionType=Cancel|refno=3HZB3605RR9VA3|rootRef=
3HZB3605RR9VA3|
```

Example of Order Replace for Multileg

```
msgType=MultiLegOrderAction|firm=ABNA|dest=STAGE|tra
nId=6|flags=4|actionType=Replace|refno=3HZB3605RR9TA3|quantity=200|orderType=
Limit|tif=DAY|category=DASHISE|price=10.02|data=2|data[0]=[security=JNJ
20120121P 5
0.000|side=B|quantity=200|instrumentType=Options|]data[1]=[security=JNJ 20120121
```

```
C 60.000|side=S|quantity=200|instrumentType=Options|]
```

Firm & User Account Creation

Setting up Firm, Users/Accounts in the back-end when they are created in the portal.

Below are the parts of the API that support the functionality.

Creating a Firm User/Account in the back-end

```
GlobalMsgDBFirmUserUpdate newUserMsg = new GlobalMsgDBFirmUserUpdate
();
newUserMsg.init();
newUserMsg.header.dest = GlobalDestination.DB;
newUserMsg.data.firm = “FIRM”;
newUserMsg.data.userid = “USERID”;
newUserMsg.data.trader = “TRADER”;
newUserMsg.data.preTradeDest = “PRERISK”
```

This will create a very basic userid entry. You choose firm/userid/trader value. As a note, trader should probably be corresponding account information.

Edit/delete firmuser

```
edit user, use GlobalMsgDBFirmUserUpdate, similar to what you do when creating a
new user, the fields that you set will override existing value in backend.
```

delete user

```
updMsg.header.flags = BaseMsg.setFlag(updMsg.header.flags,
Const_msggen.GlobalTranFlagDBDelete);
```

P&L Transaction Data

Subscribe for PLTRAN:

Example One:

```
SubscriptionData subData = new SubscriptionData();
subData.init();
subData.security = Const_msggen.GlobalWildCard;
subData.msgType = GlobalMsgType.MultiPLTranUpdate;
```

```
data.firm = firm;
data.trader = trader;
```

Example Two:

```
Subscribe for MultiPLTranUpdate, and you will receive msg MultiPLTranUpdate and
PLTranUpdate. SubscriptionData subData = new SubscriptionData();
        subData.init();
        subData.security = Const_msggen.GlobalWildCard;
        subData.msgType = GlobalMsgType.MultiPLTranUpdate;
subData.firm = firm;
subData.trader = trader;
```

You will get back GlobalMsgMultiPLTranUpdate and GlobalMsgPLTranUpdate, the difference between the two is that the first can carry a list of updates, while latter carry one update.

The following is an example:

```
msgType=MultiPLTranUpdate|subId=63|data=1|data[0]=[firm=LEHM|security=ZVZZT|t
rader=0003|pos=1000|dollarAmount=101000.0|seqNum=0|avgPrice=101.0|]
```

Cash & Position Published to PL Tran Process

Publishing positions and cash to the FI server

Below is the API to make deposit/withdrawal from FI backend.

PreTradeDest, for now, always set to “PRERISK”.

To send account update, use following API:

```
GlobalMsgAccountDepositAction actMsg = new GlobalMsgAccountDepositAction ();
      actMsg.init();
      actMsg.header.firm = firm;
      actMsg.header.dest = GlobalDestination.PRERISK;
     AccountDepositActionData action = new AccountDepositActionData ();
      action.init();
      actMsg.data.add (action);
      action.account = trader;//this is corresponding to trader field when you add user,
      action.type = GlobalAccountDepositActionType.Deposit; //or Withdraw
      action.asset = “BTC”;//let say you want to deposit 0.01 share of BTC
      action.actionAmount = 1
      action.fractionBase = 100;
```

Send above message to FI, and that is for deposit.

```
find user balance, use GlobalMsgPLTranQueryAction
```

```
GlobalMsgPLTranQueryAction nextMsg = new GlobalMsgPLTranQueryAction();
nextMsg.init();
nextMsg.header.dest = GlobalDestination.ORDSUB;
PLTranData plData = new PLTranData ();
plData.init();
plData.firm = firm;
plData.trader = trader;
plData.security = Const_msggen.GlobalWildCard;// use wildcard to query all position for this user or corresponding symbol nextMsg.data.add (plData);
```

Blockchain Wallet Registration and Deposit

There are GlobalMsgAccountDepositAction and GlobalMsgWalletRegistration for wallet related actions.

To send account update, use following API

```
GlobalMsgAccountDepositAction actMsg = new GlobalMsgAccountDepositAction ();
        actMsg.init();
        actMsg.header.firm = firm;
        actMsg.header.dest = GlobalDestination.PRERISK;
       AccountDepositActionData action = new AccountDepositActionData ();
        action.init();
        actMsg.data.add (action);
        action.account = trader;//this is corresponding to trader field when you add user,
        action.type = GlobalAccountDepositActionType.Deposit; //or Withdraw
        action.asset = “BTC”;//let say you want to deposit 0.01 share of BTC
        action.actionAmount = 1
        action.fractionBase = 100;
```

Send above Message to FI, and that is for Deposit

```
GlobalMsgWalletRegistration newMsg = new GlobalMsgWalletRegistration();
newMsg.init();
newMsg.header.dest = GlobalDestination.BCTRF;
newMsg.data.firm = firm;//firm
```

```
newMsg.data.trader = trader;//trader
newMsg.data.walletname = name;
newMsg.data.walletpass = password;
```

Deposit/Withdrawal using model AccountDepositActionData

Next fields:

- account
- type
- asset
- actionAmount
- fractionBase

Additional field from identity type of operation:

- clientID - bstring, unique identifier
- typeID - int, type of transaction operation (Secondary Market, Deal Creation, Deal Participate etc.)
- blockHash - bstring, transaction hash after Blockchain operation.

Example:

Request:

```
msgType=AccountDepositAction|firm=MSFT|dest=PRERISK|lifeTime=300|data=1|data[0]=[type=Deposit|account=OBORN|asset=COIN|fractionBase=1|actionAmount=100|id=abcdef|txTypeId=4|linkChain=Y|]
```

Response:

```
msgType=AccountDepositAction|firm=MSFT|dest=PRERISK|globalID=5120|lifeTime=300|type=DefClient|tranResponse=SUCCESS|data=1|data[0]=[type=Deposit|account=OBORN|asset=COIN|fractionBase=1|actionAmount=100|finalAmount=10200|holdAmount=0|id=abcdef|txTypeId=4|blockChainHash=0x90a179dff1c157bfc2a50d52c216bdae18b779420e4cbc0535188362736a9fd5|linkChain=Y|]
```

GlobalMsgWalletAction.

GlobalMsgWalletAction query = new GlobalMsgWalletAction ();

query.init ();

query.header.dest = GlobalDestination.BCTRF;

```
WalletActionData data = new WalletActionData ()
data.init();
data.type = GlobalWalletActionType.Query;
data.wallet = wallet;//the address of the wallet
```

To check a specific blockchain transaction, you need the blockchain hash value

```
GlobalMsgBlockChainHashQuery query = new GlobalMsgBlockChainHashQuery ();
query.init();
query.header.dest = GlobalDestination.BCTRF;
query.data.tranHash = block_hash;
```

Order Query

```
msggen.GlobalMsgDBOrderQuery query = new msggen.GlobalMsgDBOrderQuery();
query.init();
query.header.dest = msggen.GlobalDestination.DB;
query.data.whereClause = "firm = 'FIRM' AND trader='DIGI4'";
```

All Firm Wallet Query

to find all user wallets, use message GlobalMsgDBMultiFirmUserQuery, example below

```
GlobalMsgDBMultiFirmUserQuery nextMsg = new GlobalMsgDBMultiFirmUserQuery();
nextMsg.init();
nextMsg.header.dest = nextMsg.header.peerDest = GlobalDestination.DB;
nextMsg.header.firm = Const_msggen.GlobalWildCard;
nextMsg.header.flags =
BaseMsg.setFlag(nextMsg.header.flags,Const_msggen.GlobalTranFlagWaitServer);
Response is GlobalMsgDBMultiFirmUserQuery, check array list of data FirmUserData
user = result.data.get; The field safeData in FirmUserData, if valid
(BaseMsg.isFieldValid (user.safeData)) contains the wallet id for that user.
```

You may get more one one response (same message) GlobalMsgDBMultiFirmUserQuery if there are many records, to check if this is the last of response, check field BaseMsg.isFieldValid (resp.header.batchId), if batchId is valid, it means there will be more response coming back, otherwise, this is the last response.

Management of Pre-trade Risk Controls

In order to get the current pre-trade risk control settings, the following message would be used GlobalMsgDBMultiFirmUserQuery set whereClause to the criteria that you want to choose (standard SQL) from in the settings.

Request example:

```
msgType=DBMultiFirmUserQuery|firm=LEHM|dest=DB|tranId=2|whereClause=firm
=  'LEHM' and userid =  'KUN'|
```

Response example:

```
msgType=DBMultiFirmUserQuery|firm=LEHM|dest=DB|tranId=2|globalID=7243|lifeTime=
30|flags=4|type=DefClient|tranResponse=SUCCESS|where Clause=firm =  'LEHM' and
userid =  'KUN' and firm =
LEHM'|data=1|data[0]=[firm=LEHM|userid=KUN|trader=0004|user_type=O|accountType=
P|marginLimit=100000.0|stageAccessList=STAGE1|preTradeDest=PRERISK|]
```

In order to set new risk, use the message GlobalMsgDBFirmUserUpdate

Request example:

```
msgType=DBFirmUserUpdate|firm=LEHM|dest=DB|tranId=3|firm=LEHM|userid=KUN|trad
er=0004|user_type=O|accountType=P|marginLimit=20000.0|stageAccessList=STAGE1|pre
TradeDest=PRERISK|
```

Response example,

```
12:49:13.549-(TRAN[O<-25]):
msgType=DBFirmUserUpdate|tranId=3|flags=256|tranResponse=SUCCESS|
```

Risk Management Fields Description

The following is a List of db fields and corresponding screen display for the FIN Console

Database field names and screen names

- ("Firm","Firm",60),
- ("Userid","User ID",100),
- ("Totalallowqty","TotQty",90),
- ("Totalallowamount","TotAmt",90),
- ("Maxordershares","Max Ord Shares",90),
- ("Maxdollaramount","Max Ord Amt",90),
- ("Percentagelimit","Max away Pct",90),
- ("Nonmarketpercentagelimit","Non-open Max away Pct",90),
- ("Marginlimit","Margin BuyPower",90),
- ("Speedlimit", "Dup Order", 90),
- ("Speedlimit2", "Speed Check", 90),
- ("Preventnakedoptionsell", "No Naked Sell", 90),

- ("Volumepercentlimit", "ADTV %",90),
- ("Optionmaxordershares","OPT Max Ord Shares",90),
- ("Shortsale_monitor","Locate",70),
- ("Pretradedest", "Risk Dest",60),
- ("Trader","Trader",100),
- ("User_type","User Type",50),
- ("Accounttype","Account Type",60),
- ("Monitorconn","Monitor",50),
- ("Servicebureau","SveBureau",60),
- ("Status_monitor","Risk Status", 40),
- ("Attributes", "Attr", 100),
- ("Attributable","Displayable",60),
- ("Retail_type", "Extern Type", 30),
    - ("extern_account", "Extern Acct", 80),
    - ("commission", "Commission",60),
- ("Bureauid", "BureauID",60),
- ("Cancelondisconnect", "Cxl On Discon",60),
- ("Flatposafterdisconnect", "Zero Pos after Disc",60),
- ("Rolluptrader","Rollup Act", 50),
- ("Minprice","Min Price", 50),
- ("Maxprice","Max Price", 50),
- ("Maxnumpos","Max Num Pos", 50),
- ("Maxopenorders","Max Open Ords", 50),
- ("Maxgrossloss","Max Loss", 50),
- ("Riskrollup","Rollup Risk", 50),
- ("Symbolfilter","Sym Filt", 50),
- ("Concentpercentlimit","Concent Pct Lmt", 50),
- ("active_account","Active", 70),
- ("Washcheck","Acc Wash", 70),
- ("maxpossize", "Max Pos Size", 100),
- ("mastertrader", "Master Trader", 80),
- ("Instructions", "Spec Instr", 200),
- ("Oddlot_check", "Odd Lot Chk", 100),
- ("Shortsalemultiplier", "Short Multiplier", 100),
- ("Resetlimit", "Reset Limit", 100),
- ("Allow_accounts", "Sub Accts", 250),
- ("isocheck", "ISO Check", 100),
- ("Haltcheck", "HALT Check", 100),

Risk Management Notification of Rejects

If there is rejection, following strings are returned to client, most of information is straightforward.

- "Exceed max pos " //exceed max number of position
- "Exceed max open order " // exceed max open order
- "Exceeds max quantity"
- “"ISO not allowed"”
- "Cannot initiate odd lot pos"”
- "Wash sale prohibited"
- "Max dollar amount exceeded"”
- "Not in filter list" // has symbol filter and symbol is not in allowed list
- ""Exceeds margin limit"
- "Exceed total qty"
- "Exceed total Amount"

You can also subscribe for MarginInfoUpdate for information such as currentMarginlimit, buy qty, sell qty, etc…

Subscribe for margin Information:

```
subData = new SubscriptionData();
        subData.init();
        subData.firm = firm;
        subData.trader = trader;
        subData.msgType = GlobalMsgType.MarginInfoMultiUpdate;
```

Example of update message:

```
msgType=MarginInfoMultiUpdate|subId=63|data
=1|data[0]=[currentQty=200|currentAmount=2100.0|currentMargin=2100.0|firm=LEH
M|trader=0004|maxDailyQty=0|maxDailyAmount=0.0|maxMargin=20000.0|currentSp
eed=0|maxSpeed=0|currentCumNetQty=0|currentCumNetValue=0.0|currentCumOpen
Qty=200|currentCumOpenValue=2100.0|currentCumLongQty=0|currentCumLongValue
=0.0|currentCumShortQty=0|currentCumShortValue=0.0|totalOrders=2|totalTrades=0
|lastMinOrders=2|lastMinTrades=0|maxSpeed2=0|currentSpeed2=0|]
```

Risk Management for Crypto Currenacy Pairs

All crypto based currency pairs are converted to a notional dollar value.

Notional Loss limits, and other Risk Settings

- If the firm has 1 BTC and the price is $3400
- If the spot price of ETH USD is $104.00

The spot price for ETH/BTC is 030637
If the firm has 1 ETH/BTC the notional value would be .030637.00 X 3400
= **$104.1658 This would be part of notional value risk.**

Setting 'Fast Proxy Signal Light' Risk

Clients can use the message GlobalMsgOrderAction to set "green light / red light" risk instructions for inbound clients orders, it has two fields. Here is an example print out what GlobalMsgOrderAction looks like:

```
msgType=OrderAction|firm=LEHM|dest=STAGE|tranId=2|
firm=LEHM|security=ZVZZT|category=INET|actionType=Entry|side=B|price=10.5|
tif=DAY|trader=0003|quantity=1000|
```

Incoming request:

```
msgType=OrderAction|firm=LEHM|dest=STAGE|tranId=2|
firm=LEHM|security=ZVZZT|category=INET|actionType=Entry|side=B|price=10.5|
tif=DAY|trader=0003|quantity=1000|
```

If successful, response looks like this:

```
msgType=OrderAction|firm=LEHM|dest=STAGE|tranId=2
|tranResponse=SUCCESS|firm=LEHM|
security=ZVZZT|category=INET|actionType=Entry|side=B|price=10.5|
tif=DAY|trader=0003|quantity=1000
```

If unsuccessful, response looks like this:

```
14:28:10.949-(TRAN[O<-38]):
msgType=OrderAction|tranId=6|tranResponse=FAILURE|er
rorCode=NoServer|errorDetails=No match server|
```

Key fields from the request that should be returned is tranId, which the FI side uses to match up against the requests. The client can use tranResponse field to indicate success or failure. The client can set errorDetails field to give more details for the error.