OWT Trading platform

Overview

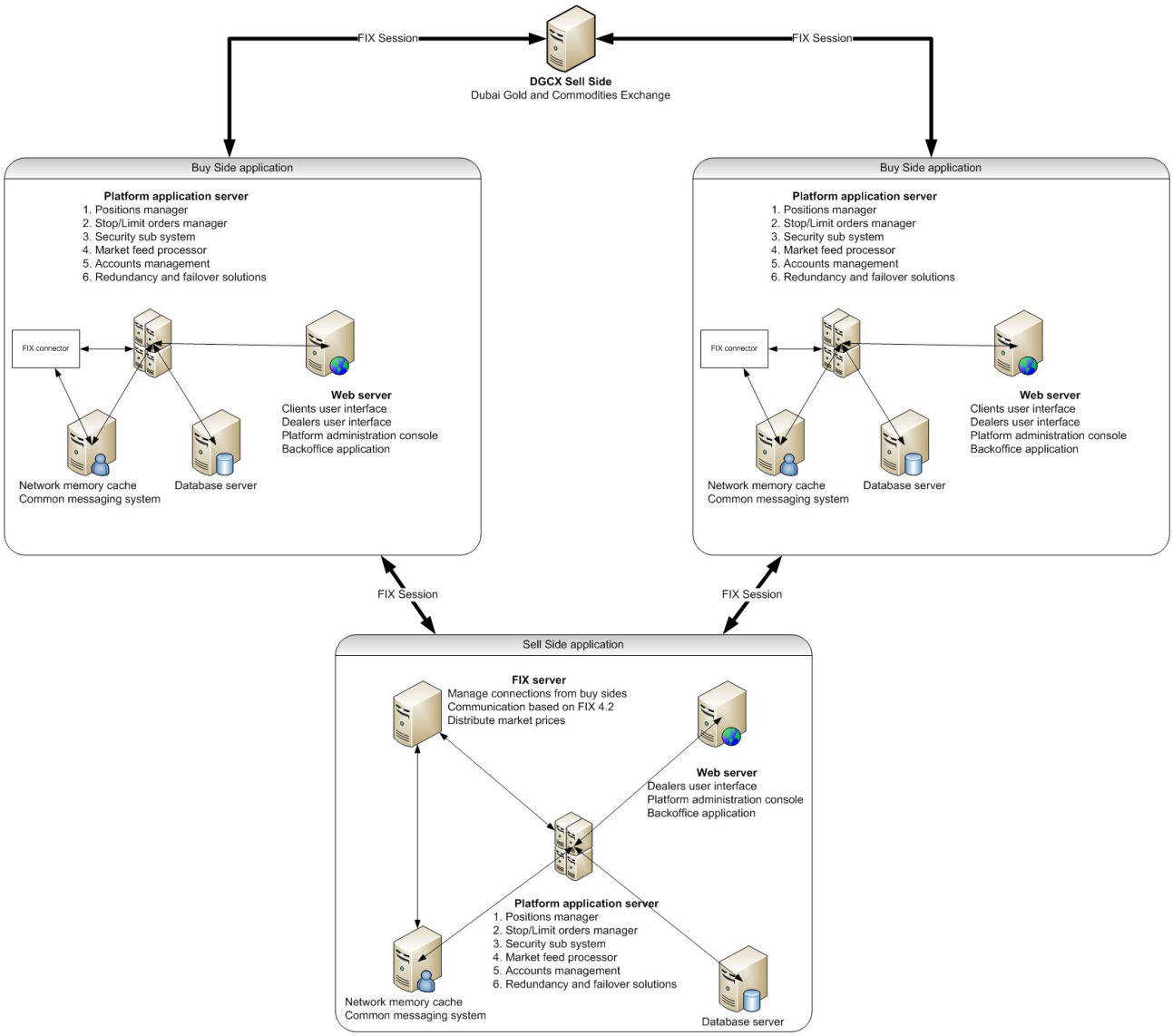
Our company was involved in designing and realization for one of our Middle East customers. Customer, very fast growing company, decided gain the market with new platform which allow clients take advantages on trading on many brokers from one application.

Developed solution was separated on 2 parts: single sell side server and numerous buy sides.

Architecture specific

Retail clients has access to their trade terminals through Web GUI provided by buy side application. This software was a combination of high level services(like **FIX** connectors, Orders managers, load balancing and failover solutions, web server and so on) and low level applications(RDBMS, network memory, real time messaging bus). Buy side has direct connection to **DGCX** market through **FIX** protocol. All FX positions requested by clients was forwarded using **FIX** 4.2 to Sell side application where they going to be processed automatically or with human intervention(depends on settings).

Sell side was designed to handle incoming **FIX** connections from Buy sides and process their requests in parallel mode. This application also has web GUI but only for dealers and administrators and almost identical set of services like Buy side has.


FIX Session
FIX Session
DGCX Sell Side
Dubai Gold and Commodities Exchange
Buy Side application
Platform application server
1. Positions manager
2. Stop/Limit orders manager
3. Security sub system
4. Market feed processor
5. Accounts management
6. Redundancy and failover solutions
FIX connector
Web server
Clients user interface
Dealers user interface
Platform administration console
Backoffice application
Network memory cache
Common messaging system
Database server
Buy Side application
Platform application server
1. Positions manager
2. Stop/Limit orders manager
3. Security sub system
4. Market feed processor
5. Accounts management
6. Redundancy and failover solutions
FIX connector
Web server
Clients user interface
Dealers user interface
Platform administration console
Backoffice application
Network memory cache
Common messaging system
Database server
FIX Session
FIX Session
Sell Side application
FIX server
Manage connections from buy sides
Communication based on FIX 4.2
Distribute market prices
Web server
Dealers user interface
Platform administration console
Backoffice application
Platform application server
1. Positions manager
2. Stop/Limit orders manager
3. Security sub system
4. Market feed processor
5. Accounts management
6. Redundancy and failover solutions
Network memory cache
Common messaging system
Database server


Status: Open new Order
Balance
Equity
Used Margin
Free Margin
Credit
Margin level

Modules for AIOTrade

At the moment you can find some projects quite good OpenSource projects intended for visualisation of the historical data with their subsequent analysis. To such we can carry and **AIOTrade**. This project is based on NetBeans Platform that, of course, affects the final

sizes of a package, however it hardly belittles all its other dignities.

Under the order of [censored] company AxonSoftware has developed a number of the additional modules expanding functionality of **AIOTrade**. Among them — real time date fetching data from remote storehouse, application for formation of storehouse of the data (does not concern to **AIOTrade** but it has been developed update the scope of the given project), visualisation and as working out of several new indicators.

Market feed transformation

Overview

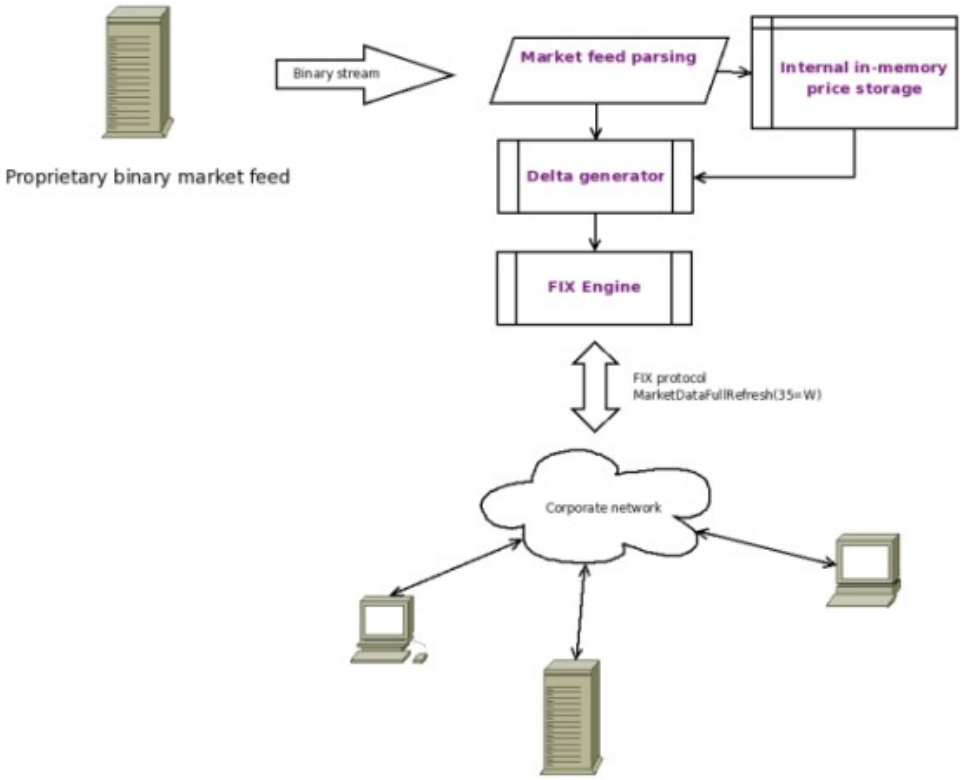
The company which has ordered given development received market price on the subscription basis. Transfer protocol used by the remote side already morally obsolete very for a long time ago. It represented binary data flow and inquiries about upgrade needed to be initiated independently. It is not so convenient and is linked to costs in case it is necessary to duplicate the data on other servers.

Our engineers has a task to develop application co-operating with a source of the market prices data feed on the one hand and with another — allowing to give to listeners/subscribers of the prices on delta model(sending goes only in case of change) under standard protocol.

Architecture specific

As the result has been created application consisting of several independent and nevertheless co-operating with the each other components. Thread with high priority fulfilled connection with original market feed and spent reading. Results were held in in memory cache. When changes was detected data was transferred for the subsequent processing to a component, responsible multicasting. Given component represented simplified **FIX** server making dispatch of upgrades of the prices to all subscribers (using message MarketDataFullRefresh). All data was transferred on the encrypted protected channel.

Given application has allowed to simplify considerably data transfer on other hosts, usage of **FIX** protocol has facilitated further development. We considered data transfer in **FAST**, however within the limits of the given task the quantity of the transferred data was not so high the same as also number of subscribers (~ 10).

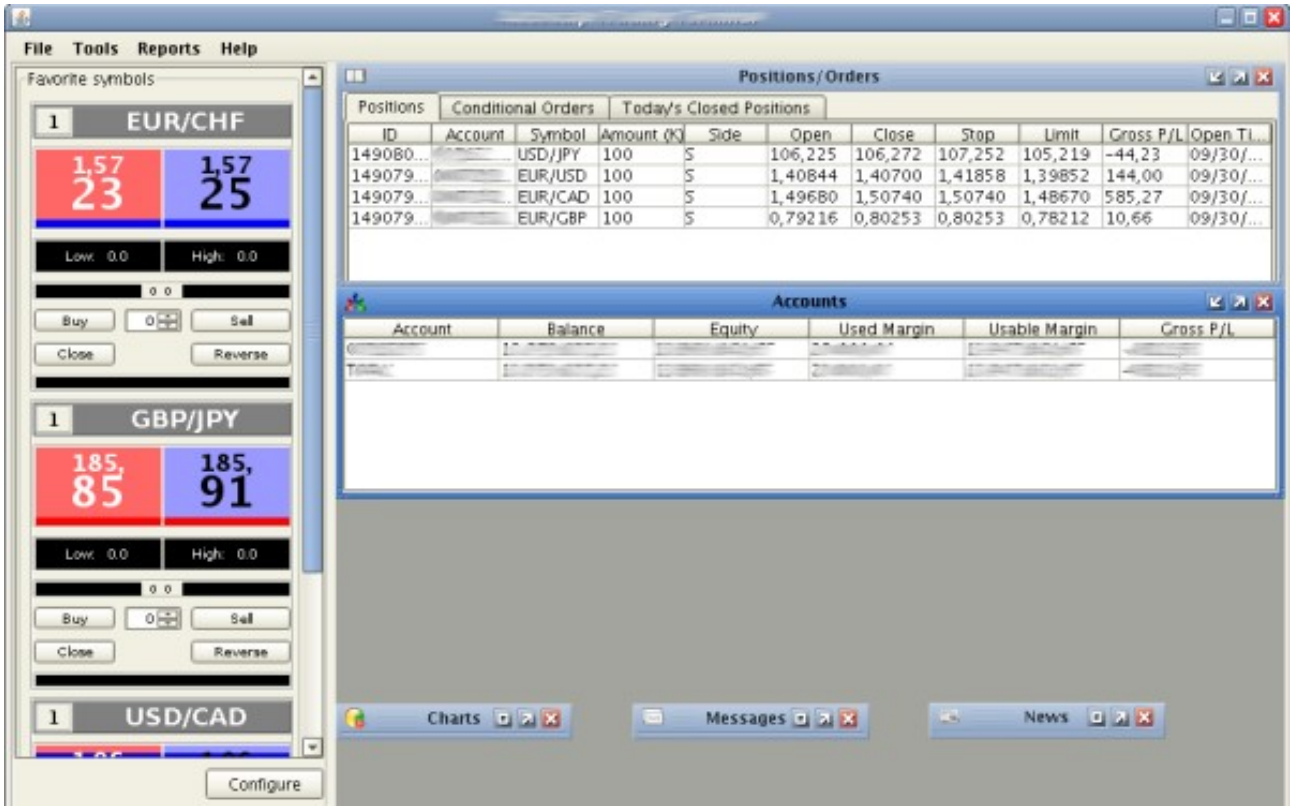

Proprietary binary market feed
Binary stream
Market feed parsing
Internal in-memory price storage
Delta generator
FIX Engine
FIX protocol
MarketDataFullRefresh(35=W)
Corporate network

Trading terminal framework

Overview

Under scope of several projects our engineers was involved in development of trading terminals. This application should not be most powerful in the class. Minimum size, simplicity of usage, reliability, cross-platform - this was the main requirements. Terminal should be used inside organisation allow to perform some manual trading operations: open/close positions, to place/change/cancel conditional orders, view news and messages, generate reports.

As a result our engineers developed a framework which in a consequence was changed several times under needs of the concrete client. Java has been chosen as programming language that at once has dealt with a question with portability. Depending on the project terminal could will be connected directly to FIX server(in this case all messages were transferred with FIX 4.4), or through various messaging systems.



Distributed historical price storage

Overview

FIX Connector is interface to selected market feed provider. He[Connector] responsible for any interactions through FIX protocol with counterpart - login/logout, market feed subscription, receiving and decoding real time price ticks, handling network problems like disconnections. Each received price tick forwarded to **PriceWriter**, part, responsible for saving information in local database.

All communications with end clients are moved to **Application Server**(AS). This is separate program connected to historical database. **AS** provide simple authentications mechanisms for new incoming connections and several application-related functions like ability retrieve list of all available symbols and timeframes, get historical data. We storing in database raw price ticks which allow **AS** build quotes according to client's request. Calculated result will be streamed to client using very simple protocol.

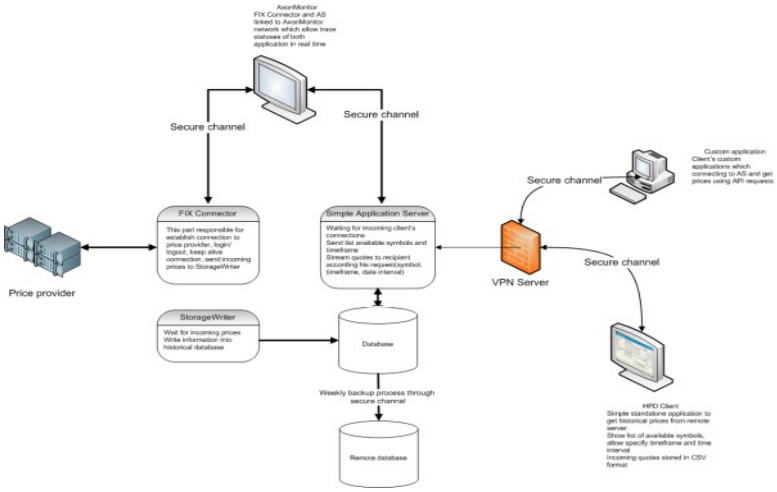
To simplify import of historical data in third-party applications we developed small graphical tool which talking with **AS** using API and store results in CSV files.

Maintenance

Such type of applications required attention from the side of people. To prevent lost of data we organized weekly backups to remote database. **FIX Connector** and **Application Server** linked to our AxonMonitor solution so engineers of our company or customer stuff can trace states of both applications in real time.

Architecture specific

Separation of given application on two parts was necessary step. In case **FIX Connector** is crushed, client still have ability get historical data and otherwise - if something wrong with **AS**, **FIX Connector** continues storing information in database. Of course, situation, when both applications is not working is similarly possible, but improbable. Also we have ability very easy change price provider. **Application Server** working directly with data, stored in database so you can connect any type of pipeline to push new prices in DB.


AxonMonitor
FIX Connector and AS linked to AxonMonitor network which allow trace statuses of both application in real time
Secure channel
Secure channel
Secure channel
Custom application
Client's custom applications which connecting to AS and get prices using API requests
FIX Connector
This part responsible for establish connection to price provider, login/ logout, keep alive connection, send incoming prices to StorageWriter
Simple Application Server
Waiting for incoming client's connections
Send list available symbols and timeframe
Stream quotes to recipient according his request(symbol, timeframe, data interval)
Price provider
VPN Server
Secure channel
StorageWriter
Wait for incoming prices
Write information into historical database
Database
Weekly backup process through secure channel
Remote database
HPD Client
Simple standalone application to get historical prices from remote server
Show list of available symbols, allow specify timeframe and time interval
Incoming quotes stored in CSV format

Axon FIX Debugger

Overview

Engineers of our company constantly involving in integration development and constantly collided with greater time losses at debugging **FIX** components of application. The received experience has poured out in realization of the program under name **Axon FIX Debugger**, intended for simulation of behaviour of sell side. What advantages can be received using **Axon FIX Debugger**? Here some of them:

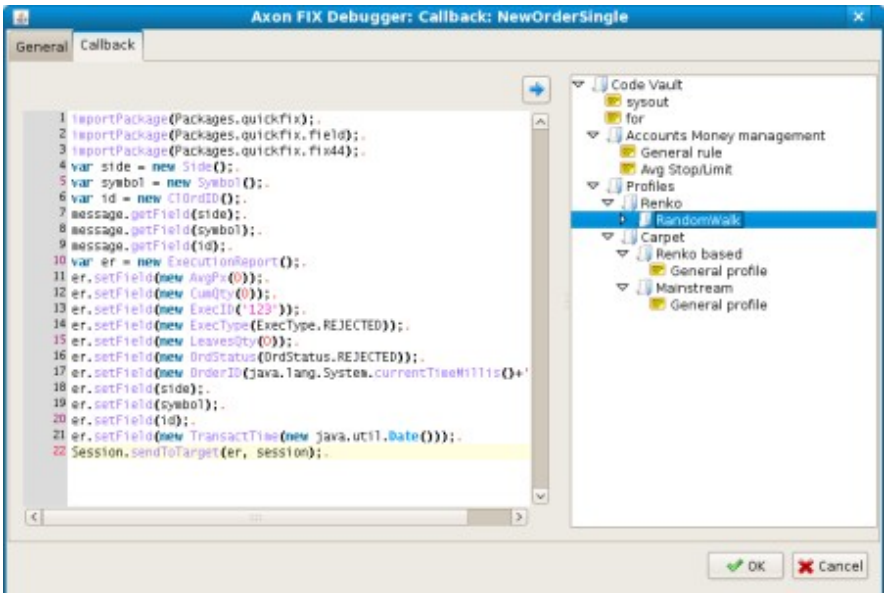
- ability to simulate work of sell side on the basis of versions 4.0, 4.2, 4.4 of **FIX** protocol
- ability independently form messages which will be sent to your application
- ability investigate trading session in a graphic kind. Debugger will give you the list of entering and proceeding messages with breakdown on fields
- built-in programming language, allowing to set as much as flexible answers on messages coming from your side

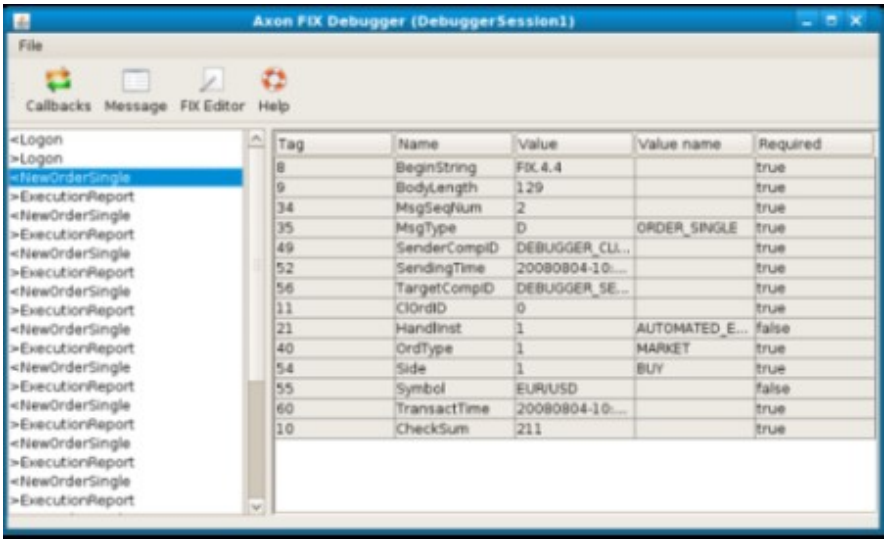
Main principles of work

Axon FIX Debugger are tiny sell side application with the graphic console and some additional opportunities.At start the debugger creates session and expects connection of the client. As soon as session is established, the client application can start send **FIX** messages. Received messages will be displayed on the screen. The debugger will allow you to familiarize with contents of each of them in more details.
Developer can independent, at any moment, generate any message and send it in the debugged application. For interactive interaction the opportunity of the task callback - functions which will be called by debugger when client will send **FIX** message which we looking for.

Basic features of **Axon FIX Debugger**

- Quantity of simultaneous client sessions: one
- Graphic debugging panel, allowing to study messages
- Sending the messages to the debugged application in a manual mode
- Ability to bind callback functions on any entering **FIX** message
- Built-in programming language, allowing to form answers of any complexity
- Editor with syntax highlighting and opportunity to include created patterns of a code (code snippets)
- Delivery with a standard set of callbacks
- Built-in help system describing the basic functions of the programming language and containing examples of their usage
- Editor, allowing to make changes in standard **FIX** protocol - enter new fields and values
- Multiplatform. **Axon FIX Debugger** it is capable to work on Linux, Win32 and MacOS.


Axon FIX Debugger: Callback: NewOrderSingle
General
Callback
importPackage(Packages.quickfix);
importPackage(Packages.quickfix.field);
importPackage(Packages.quickfix.fix44);
var side = new Side();
var symbol = new Symbol();
var id = new ClOrdID();
message.getField(side);
message.getField(symbol);
message.getField(id);
var er = new ExecutionReport();
er.setField(new AvgPx(0));
er.setField(new CumQty(0));
er.setField(new ExecID("123"));
er.setField(new ExecType(ExecType.REJECTED));
er.setField(new LeavesQty(0));
er.setField(new OrdStatus(OrdStatus.REJECTED));
er.setField(new OrderID(java.lang.System.currentTimeMillis()+"
er.setField(side);
er.setField(symbol);
er.setField(id);
er.setField(new TransactTime(new java.util.Date()));
Session.sendToTarget(er, session);
Code Vault
sysout
for
Accounts Money management
General rule
Avg Stop/Limit
Profiles
Renko
RandomWalk
Carpet
Renko based
General profile
Mainstream
General profile
OK
Cancel


Axon FIX Debugger (DebuggerSession1)
File
Callbacks
Message
FIX Editor
Help
<Logon
>Logon
<NewOrderSingle
>ExecutionReport
<NewOrderSingle
>ExecutionReport
<NewOrderSingle
>ExecutionReport
<NewOrderSingle
>ExecutionReport
<NewOrderSingle
>ExecutionReport
<NewOrderSingle
>ExecutionReport
<NewOrderSingle
>ExecutionReport
<NewOrderSingle
>ExecutionReport
<NewOrderSingle
>ExecutionReport
<NewOrderSingle
>ExecutionReport
Tag | Name | Value | Value name | Required
8 | BeginString | FIX.4.4 | | true
9 | BodyLength | 129 | | true
34 | MsgSeqNum | 2 | | true
35 | MsgType | D | ORDER_SINGLE | true
49 | SenderCompID | DEBUGGER_CLI... | | true
52 | SendingTime | 20080804-10:... | | true
56 | TargetCompID | DEBUGGER_SE... | | true
11 | ClOrdID | 0 | | true
21 | HandlInst | 1 | AUTOMATED_E... | false
40 | OrdType | 1 | MARKET | true
54 | Side | 1 | BUY | true
55 | Symbol | EUR/USD | | false
60 | TransactTime | 20080804-10:... | | true
10 | CheckSum | 211 | | true