FIXimulator: A Financial Information eXchange Protocol Compliant Sell Side Trading Application

Zoltan Feledy

A Thesis in the Field of Information Technology

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

February 2009

Abstract

FIXimulator is a Java implementation of a trading application used to communicate transactions with trading partners through a widely used protocol called FIX. When running in an automated mode, the application generates random trading responses thereby simulating actions of trading partners. When used in a manual mode FIXimulator serves as a regular trading tool. FIXimulator is very flexible in the formation of FIX messages and supports the actions in many different sell-side trading situations. The target audience for this application is anyone interested in the development and testing of trading applications that make use of the FIX protocol. This work demonstrates the feasibility of Java implementations of trading systems using the FIX protocol.

Acknowledgments

I would like to thank my thesis director, Dr. Zoran Djordjevic, for his guidance throughout this project.

I would also like to thank Dr. Bill Robinson for all his help during the proposal stage of this project and for all his efforts and patience during the process of finding a director for this thesis.

I want to extend special thanks to Dr. Jeff Parker for his thorough review of the text and for the many clarifications.

Special thanks to Amy Timney for generously volunteering her time for proofreading and editing this document.

Many thanks to my wife, Yulia, and to my family for their understanding, support and encouragement without which this project would not have come to fruition.

Table of Contents

List of Figures

Chapter 1 Introduction

The financial industry is often spoken of as having two sides, the buy side and the sell side. The buy side players are generally asset managers who trade various financial instruments on behalf of their clients or themselves. They manage portfolios of different assets and trade those assets according to various portfolio management guidelines. Their trading partners are the sell side firms which help the buy side by locating trading partners for their trades. The sell side is also known as the broker/dealer community.

In the early days of the stock market and other financial markets, trades were communicated over the telephone to traders working on the floors of the exchanges. It was not until the 1980's that the industry began investing heavily into computers to automate trade related communication. At the time communicating trade information electronically was the cutting edge technology used only by the largest firms on Wall Street. They used electronic trading to differentiate themselves and to gain competitive advantage. The next fifteen years were spent developing a number of proprietary trade protocols that became increasingly difficult to maintain due to the lack of standards. In the early 90's Fidelity decided to publish the communication framework that they were using with Salomon Brothers to trade equities which formed the foundation of FIX (FIX Protocol Limited, 2007).

FIX 2.7 was published in early 1995, quickly followed by FIX 3.0. Neither one of these versions was widely implemented. The first stable version, FIX 4.0, was released in 1996. By this time FPL (FIX Protocol Limited), the organization responsible for maintaining the protocol, and its mission was clearly established. There are still many production lines in use today (2008) that utilize the 4.0 version of the protocol. FIX 4.0 had many ambiguities and only supported equity trading. Many of the issues were addressed in FIX 4.1 but only a few firms adopted this version. FIX 4.2 was released in 2000 with corrections in 2001 and is probably the most widely used version in production today for equity trading. FIX 4.3 was published in 2001 to add support for fixed income securities and FIX 4.4 in 2003 added support for post trade processing. Upgrading production systems with these versions as the FIX protocol expanded was becoming increasingly difficult. A new idea of separating session and application level messaging was conceived in 2006 to solve this problem. Session level messaging was separated out to allow multiple versions of application messages to be transported on the same session. This new session level protocol was named FIXT.1.1 and it facilitates the transport of multiple versions on the same session. FIX 5.0 is the latest version that was released in October 2006 with good support for foreign exchange trading and most importantly, support for the transport independent framework (FIX Protocol Limited, 2007).

FIX is a communication protocol that is quite comprehensive and difficult to implement. There are many commercial implementations that are generally very costly. There are also a small number of open source implementations. QuickFIX was the original implementation written in C++ with APIs for a number of different languages (QuickFIX, 2001). QuickFIX/J evolved from QuickFIX and is a pure Java implementation (QuickFIX/J, 2006).

Trading applications that use the FIX protocol to serve market participants are generally very complicated enterprise applications. Implementing these commercial trading platforms is a very costly and time consuming undertaking often requiring several years to implement. They further require constant maintenance and enhancements as business requirements change with market development, regulation, and other forces affecting the trading industry. Testing these applications require a way to simulate the actions of the opposite side. Many commercial simulators exist to meet these needs that are also costly and difficult to learn as analysts do not often possess the expertise of the other side. Open source applications would offer an ideal solution to address this need. Both QuickFIX projects offer a very simple buy-side application called Banzai that is used as an example application. A more sophisticated buy-side project called Marketcetera is also available however there are no open source projects for the sell side.

My project, FIXimulator, addresses the need for an open source sell side application implementing a recent version of the FIX protocol. FIXimulator uses the QuickFIX/J implementation of the FIX protocol. FIXimulator itself is written entirely in the Java programming language and it uses the Java Swing libraries for its graphical user interface. It supports the sell side actions in a general equity trading workflow. FIXimulator uses the 4.2 version of the specification since this version provides good support for equity trading and is widely used in the industry. A basic requirement for this project is demonstrating the feasibility of a Java implementation. Since this is not a commercial product and the scope of a thesis project is necessarily limited the resulting code is not thoroughly tested in the sense of verifying its ability to carry through all possible trading scenarios. In the case of the large number of most common scenarios that were tested the application performed correctly.

In the lifecycle of an equity trade, an order to buy or sell a given stock can go through a number of different states. When an order is placed with a counterparty, it may be accepted or rejected, and later may be executed or cancelled. Each of these scenarios, along with many others, is carefully specified in the protocol and is collected in a comprehensive list of order state changes. This list can be found in Appendix D of the FIX 4.2 Specification (FIX Protocol Limited, 2001) and it has been used by the industry to convert these state changes into application level test cases. These cases begin with instances in which a simple order is filled completely and extend into complicated scenarios in which orders through a number of revisions. Most of these are supported by FIXimulator and many are listed in the test cases in Appendix 1

To help the reader of this text understand the intricacies of the communications involved in typical trades we are presenting a brief description of two common scenarios or workflows as they are referred to in the trade.

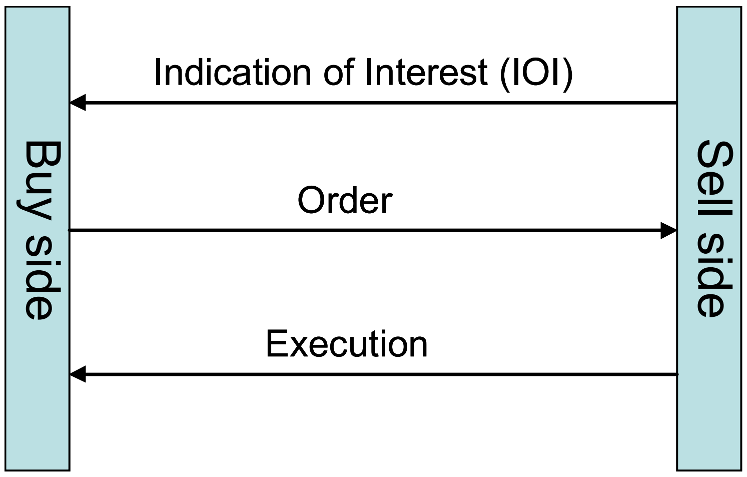


Figure 1 Sample workflow diagram for an equity transaction

In the workflow shown in Figure 1, the sell side sends an IOI (Indication of Interest) to the buy side, which is essentially an advertisement for a trade. An IOI contains information about a possible order that a sell side firm may have in its order book. As an example, an IOI could indicate that there are 100,000 shares of IBM for sale at a certain price. Should a buy side firm be looking for buying opportunities in the advertised security, it may place an order to buy some number of shares of that security. Let's assume that the buy side is interested in purchasing 5,000 shares of IBM, and it places an order for those shares. At this point, the sell side has an active order, and it has to act on it. It will then either execute the order from its order book or find a trading partner to execute the trade with. When it locates the counterparty it will report the transaction back to the buy side. A notification of a completed transaction is called an execution.

The above is one of the simplest trading workflows. The flexibility inherent in trading allows for many more complicated workflows. Appendix D of the FIX 4.2 specification contains 37 order state matrices that are achievable within trading workflows (FIX Protocol Limited, 2001). These matrices are often used in industry to create test plans for testing trade communications between counterparties. FIXimulator is able to perform most of the actions required by the sell side to satisfy the above state changes from the FIX specification therefore it is suitable for testing many enterprise applications. At this stage, FIXimulator is a demonstration project and not a commercial application and we have not completed verification of its performance in all possible trading scenarios. Test case D17 from the specification is an example of a more complicated workflow. A diagram of this workflow is shown in Figure 2.

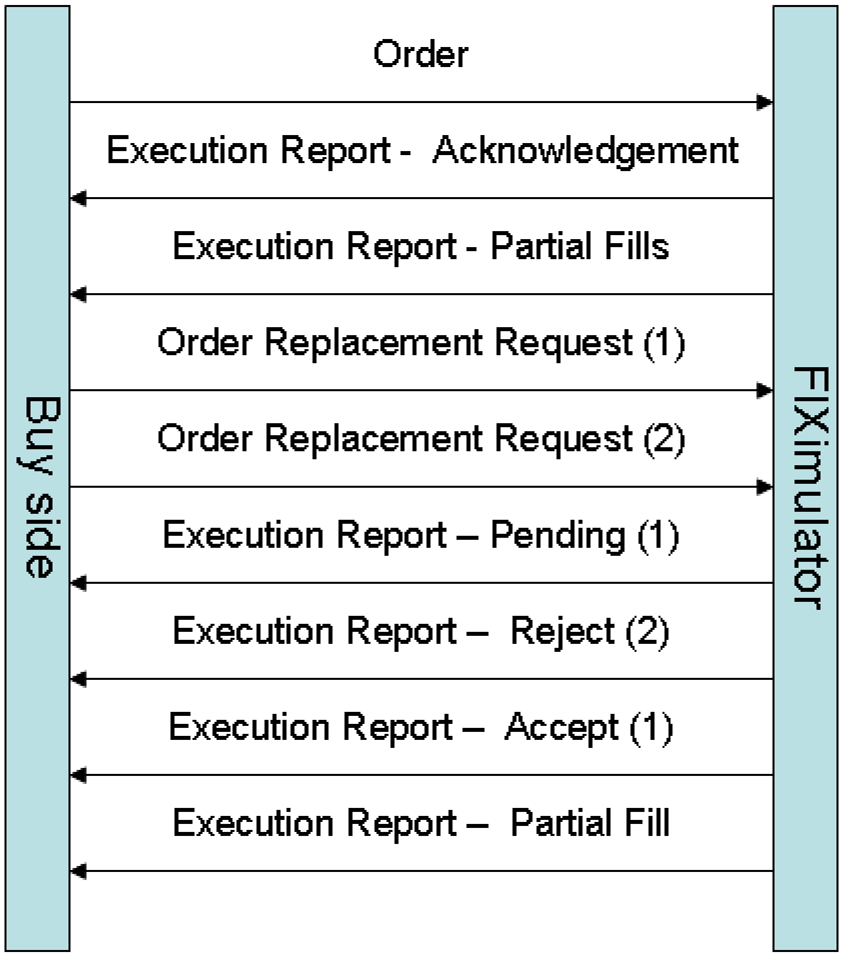

Buy side
FIXimulator
Order
Execution Report - Acknowledgement
Execution Report - Partial Fills
Order Replacement Request (1)
Order Replacement Request (2)
Execution Report – Pending (1)
Execution Report – Reject (2)
Execution Report – Accept (1)
Execution Report – Partial Fill

Figure 2 Workflow diagram for case D17 of Appendix D of the FIX 4.2 Specification

The case outlined in Figure 2 deals with the chaining of multiple requests for replacement. The description of this case reads “One cancel/replace request is issued followed immediately by another – broker rejects the second as order is pending replace.” There are nine messages required for this exchange, six of which are required by the sell side. All are supported by FIXimulator.

Chapter 2 describes the FIX protocol and the QuickFIX/J FIX engine. Chapter 3 discusses FIXimulator’s design and architecture. Chapter 4 provides an installation guide which is followed by a user guide in Chapter 5 The last chapter summarizes the project and provides a brief analysis. A glossary and a references section can be found after the last chapter along with 2 appendices. Appendix 1 contains the use cases, some of which were derived from the FIX 4.2 Specification. Appendix 2 has the application code organized by languages and packages.

Chapter 2 FIX Protocol and QuickFIX/J Engine

FIX Protocol

FIX protocol is a messaging standard that was created to automate trading between counterparties. It is a communication protocol that uses tag and value pairs. Each FIX tag is assigned both a name and a number. The pairs are separated by an ASCII null character to form messages. In various textual representations and log files this null character is frequently replaced by a readable character such as "|" to make messages easier to read. This document will follow this convention. The FIX 4.2 version lists 446 tags that can be used to assemble messages. These tags can be used to create 46 different messages that are defined as either session or application messages (FIX Protocol Limited, 2001). In actual messages, tags are represented by their numeric codes. When describing the use of various tags, this document will follow the <descriptive tag name> (<tag number>) pattern when referencing FIX tags.

Every FIX message is identified by the MsgType (35) tag which is a required field in every FIX message. This tells receiving applications what the purpose of the message is so that the messages can be appropriately handled. Of the 46 different messages that can be created in the FIX 4.2 version there are 7 that are defined as session level messages. These messages ensure that a correct session is established and kept alive. They also provide mechanisms that ensure that messages are delivered and can be recovered. In every session, it is previously agreed that one party is going to initiate a connection while another will accept an incoming connection. Once both applications are running, a session is initiated by sending a logon message as follows.

```
8=FIX.4.2|9=72|35=A|34=1|49=BANZAI|52=20081005-
14:32:42.042|56=FIXIMULATOR|98=0|108=30|10=252|
```

The above logon message as indicated by the third tag in the message, MsgType (35) being set to the value "A" is a session level message. The example is a single line from a log file. It appears on 2 lines due to a space constraint that causes the line to break at the SendingTime (52) field "`52=20081005-14:32:42.042|`."

Each message is broken down into 3 parts, a standard message header, a body and a standard message trailer. In the above example, the first 7 tag and value pairs, beginning with "`8=FIX.4.2`..." and ending with "...`56=FIXIMULATOR`", constitute the header and they are strictly specified. The first tag BeginString (8) is always the first field in a FIX message and is always unencrypted. This tells the receiving party what version of the protocol is being used. The next tag in the message, BodyLength (9), is always the second field in the message. Its value, 72, is the number of bytes contained in the message starting with the next tag, "`35=A`..." up to and including the null character before the last tag CheckSum (10) is reached, "...`108=30|`." In the above example there are 72 bytes of data which correspond to the number of characters. As previously mentioned, the next pair, "`35=A`" indicates that this is a logon message and is always the third field in a FIX message. All FIX messages are also numbered and the next tag MsgSeqNum (34) contains the sequence number of this message, in this case 1. SenderCompID (49) identifies the sender of the message, BANZAI, and TargetCompID (56) identifies the recipient, FIXIMULATOR, both of these fields are required. BANZAI is a sample buy-side application that is part of the QuickFIX/J libraries that was used to test FIXimulator. Between these two fields is SendingTime (52) which is also a required field and in the above case concludes the header section.

The body of a Logon (A) message is quite simple, here only containing two tags. EncryptMethod (98) being set to “0” simply says that this message is not encrypted and HeartBtInt (108) being set to “30” indicates that in periods of inactivity a Heartbeat (0) message will be sent every 30 seconds to keep the session alive. Heartbeat (0) refers to a Heartbeat message where MsgType (35) would be set to “0” to indicate that it is a heartbeat message which is different from the tag descriptions previously used.

The last tag in the message CheckSum (10) constitutes the trailer. This field is always the last field in a message and is always unencrypted. It serves as the end of message delimiter and it contains a 3 byte simple checksum, meaning that it will always have 3 characters. As a result, a FIX message will always end with a “10=###|” pattern. The calculation of the checksum is specified in Appendix B of the FIX specification.

The above message translated to English would read as follows. “This is a FIX 4.2 message containing 72 characters before the checksum field. It is a logon message to initiate a FIX session with sequence number 1. It is being sent to FIXIMULATOR from BANZAI at 14:32:42.042 UTC on October 5, 2008. This message is not encrypted, and during this session heartbeat messages will be sent every 30 seconds. The checksum of this message is 252.”

There are only 7 types of session level messages, Heartbeat (0), Logon (A), Logout (5), Reject (3), Resend Request (2), Sequence Reset (4) and Test Request (1). These 7 messages ensure that a connection on which application messages can be sent is in place and functioning properly. The FIX 4.2 specification lists 39 different application level messages. These would include messages such as Indication of Interest (6), New Order – Single (D), and Execution Report (8).

On the next page is a sample of what a FIX session would look like if the only trading activity would be what is described in Figure 1. Some Heartbeat (0) messages were removed for brevity. Here is what is depicted in the following session:

Message	SenderCompID (49)	MsgType (35)	TargetCompID (56)
1	BANZAI	Logon (A)	FIXIMULATOR
2	FIXIMULATOR	Logon (A)	BANZAI
3	FIXIMULATOR	Heartbeat (0)	BANZAI
4	BANZAI	Heartbeat (0)	FIXIMULATOR
5	BANZAI	Heartbeat (0)	FIXIMULATOR
6	FIXIMULATOR	Heartbeat (0)	BANZAI
<Heartbeat (0) messages were removed>			
7	FIXIMULATOR	Indication of Interest (6)	BANZAI
<Heartbeat (0) messages were removed>			
8	BANZAI	New Order – Single (D)	FIXIMULATOR
9	FIXIMULATOR	Heartbeat (0)	BANZAI
10	FIXIMULATOR	Execution Report (8)	BANZAI
<Heartbeat (0) messages were removed>			
11	FIXIMULATOR	Execution Report (8)	BANZAI
<Heartbeat (0) messages were removed>			
12	BANZAI	Logout (5)	FIXIMULATOR

Each FIX message begins with "`8=FIX.4.2`..." and ends with "...`10=252|`." They always appear on single lines in log files however some of the longer messages on the next page take up multiple lines due to space limitations. Following the log section each message is individually discussed.

```
8=FIX.4.2|9=72|35=A|34=1|49=BANZAI|52=20081005-14:32:42.042|56=FIXIMULATOR|98=0|108=30|10=252|
8=FIX.4.2|9=72|35=A|34=1|49=FIXIMULATOR|52=20081005-14:32:42.292|56=BANZAI|98=0|108=30|10=003|
8=FIX.4.2|9=60|35=0|34=2|49=FIXIMULATOR|52=20081005-14:33:12.462|56=BANZAI|10=212|
8=FIX.4.2|9=60|35=0|34=2|49=BANZAI|52=20081005-14:33:12.525|56=FIXIMULATOR|10=212|
8=FIX.4.2|9=60|35=0|34=3|49=BANZAI|52=20081005-14:33:43.007|56=FIXIMULATOR|10=212|
8=FIX.4.2|9=60|35=0|34=3|49=FIXIMULATOR|52=20081005-14:33:43.069|56=BANZAI|10=220|
.
.
.
8=FIX.4.2|9=198|35=6|34=5|49=FIXIMULATOR|52=20081005-
14:34:35.678|56=BANZAI|15=USD|22=5|23=I1223217185850|27=100000|28=N|44=103.5|48=IBM.N|54=1|55=IBM|62=200810
05-15:04:35.677|107=INTL BUSINESS MACHINES CORP|130=Y|10=036|
.
.
.
8=FIX.4.2|9=137|35=D|34=8|49=BANZAI|52=20081005-
14:35:46.672|56=FIXIMULATOR|11=1223217346597|21=1|38=5000|40=1|54=2|55=IBM|59=0|60=20081005-
14:35:46.668|10=153|
8=FIX.4.2|9=60|35=0|34=8|49=FIXIMULATOR|52=20081005-14:36:07.419|56=BANZAI|10=227|
8=FIX.4.2|9=177|35=8|34=9|49=FIXIMULATOR|52=20081005-
14:36:07.512|56=BANZAI|6=0|11=1223217346597|14=0|17=E1223217367463|20=0|31=0|32=0|37=O1223217347023|38=5000
|39=0|54=2|55=IBM|150=0|151=5000|10=164|
.
.
.
8=FIX.4.2|9=189|35=8|34=11|49=FIXIMULATOR|52=20081005-
14:36:43.201|56=BANZAI|6=103.5|11=1223217346597|14=5000|17=E1223217373812|20=0|31=103.5|32=5000|37=O1223217
347023|38=5000|39=2|54=2|55=IBM|150=2|151=0|10=237|
.
.
.
8=FIX.4.2|9=61|35=5|34=17|49=BANZAI|52=20081005-14:40:00.014|56=FIXIMULATOR|10=004|
```

The first message in the above log is a Logon (A) message initiated by BANZAI as in this example it was previously agreed that BANZAI will initiate the connection. FIXIMULATOR responded with a Logon (A) message indicating that the logon was accepted. After each 30 seconds of activity Heartbeat (0) messages are exchanged. Either side can initiate these heartbeats after 30 seconds of inactivity. In the above case the first heartbeat exchange is initiated by FIXIMULATOR as BANZAI had to process the logon response so the 30 seconds of inactivity first expired for FIXIMULATOR.

Following some exchanges of Heartbeat (0) messages an IOI is sent by the sell side, in this case FIXIMULATOR.

```
8=FIX.4.2|9=198|35=6|34=5|49=FIXIMULATOR|52=20081005-
14:34:35.678|56=BANZAI|15=USD|22=5|23=I1223217185850|27=100000|28=N|44=
103.5|48=IBM.N|54=1|55=IBM|62=20081005-15:04:35.677|107=INTL BUSINESS
MACHINES CORP|130=Y|10=036|
```

In the example above, this message indicates a buyer for a 100,000 shares of IBM at the price of 103.50. “35=6” indicates that it is an IOI message. IOIShares (27) indicates that 100,000 shares are being advertised at a price specified in Price (44). For this example the buy side responded by ordering 5,000 shares because they had an order for that quantity in their order book.

```
8=FIX.4.2|9=137|35=D|34=8|49=BANZAI|52=20081005-
14:35:46.672|56=FIXIMULATOR|11=1223217346597|21=1|38=5000|40=1|54=2|55=
IBM|59=0|60=20081005-14:35:46.668|10=153|
```

OrderQty (38) specifies the number of shares, Side (54) set to 2 implies that this is a sell transaction of the security specified by a ticker symbol in this case “IBM” in the Symbol (55) field. Separated by an exchange of Heartbeat (0) messages, there are 2 Execution Report (8) messages as indicated by the MsgType (35) field being set to “8”.

```
8=FIX.4.2|9=177|35=8|34=9|49=FIXIMULATOR|52=20081005-
14:36:07.512|56=BANZAI|6=0|11=1223217346597|14=0|17=E1223217367463|20=0
```

```
|31=0|32=0|37=O1223217347023|38=5000|39=0|54=2|55=IBM|150=0|151=5000|10
=164|
8=FIX.4.2|9=60|35=0|34=9|49=BANZAI|52=20081005-
14:36:16.976|56=FIXIMULATOR|10=236|
8=FIX.4.2|9=61|35=0|34=10|49=FIXIMULATOR|52=20081005-
14:36:38.420|56=BANZAI|10=009|
8=FIX.4.2|9=189|35=8|34=11|49=FIXIMULATOR|52=20081005-
14:36:43.201|56=BANZAI|6=103.5|11=1223217346597|14=5000|17=E12232173738
12|20=0|31=103.5|32=5000|37=O1223217347023|38=5000|39=2|54=2|55=IBM|150
=2|151=0|10=237|
```

The first of these two is called an "acknowledgement" which simply indicates the receipt of the order. The status of an order can be tracked by the OrdStatus (39) field. An acknowledgement is achieved by setting the OrdStatus (39) field to "0" to indicate the order status of "New". The second Execution Report (8) fills the order as indicated by the OrdStatus (39) field being set to "2" to indicate the status as "Filled". The price and the quantity of the fill are indicated in LastShares (32) and LastPx (31). In the above case 5,000 were sold at a price of 103.5 as originally advertised in the IOI. To show a complete FIX session there are some more exchanges of Heartbeat (0) messages terminated by a Logout (5) message.

```
8=FIX.4.2|9=61|35=5|34=17|49=BANZAI|52=20081005-
14:40:00.014|56=FIXIMULATOR|10=004|
```

Where a Logon (A) message requires a response, a Logout (5) message does not. Since it is previously agreed and configured that BANZAI is acting as the initiator, a session is disconnected when a Logout (5) message is received. Once this message is received, the FIX session is terminated.

There are a number of commercial implementations of the FIX protocol as well as two very closely related open source implementations – QuickFIX and QuickFIX/J. The FIX website currently lists over 30 commercial offerings. These session level implementations are known as FIX engines and they all offer an API to create applications and application level messaging (FIX Protocol Limited, 2008). FIXimulator is built using the open source Java implementation of the protocol, QuickFIX/J. This was a good fit for this project because it is free and will allow FIXimulator to also be offered for free as an open source application. In particular, FIXimulator could readily be modified by an interested user, expanded or adjusted to new requirements. Such freedom does not exist with commercial applications.

QuickFIX/J FIX Engine

QuickFIX/J is an open source FIX engine that addresses the session level messaging of the FIX protocol thereby providing a foundation for sophisticated applications. Once a session is configured, QuickFIX/J establishes each connection and keeps it alive. The application programmer's only responsibility is to create and handle application messages as seen in the previous section. The Logon (A), Heartbeat (0), and Logout (5) messages are automatically generated along with all the other session level messages. The application programmer therefore is only responsible for application messages like the Indication of Interest (6) and the Execution Report (8) messages above. QuickFIX/J offers full support of FIX 4.0 to FIX 4.4 versions of the protocol (QuickFIX/J, 2008) and since FIXimulator uses the 4.2 version, it is a suitable engine for this project.

QuickFIX/J also offers multiple persistence and logging options including console based logging for the highest performance as well as flat file and database logging for permanent persistence (QuickFIX/J, 2008). Along with these options, logging to multiple destinations is also supported by QuickFIX/J. FIXimulator utilizes these features to provide flexibility in the selection of logging options to the user.

Chapter 3 Design and Architecture

FIXimulator consists of several modules:

1. FIXimulator Module
    a. Core application
    b. User interface
    c. Utilities
2. The QuickFIX/J FIX engine
3. MySQL Database
4. Configuration files

Figure 3 shows a top level architectural diagram and how the above components communicate with each other and ultimately with buy-side counterparties through a network.

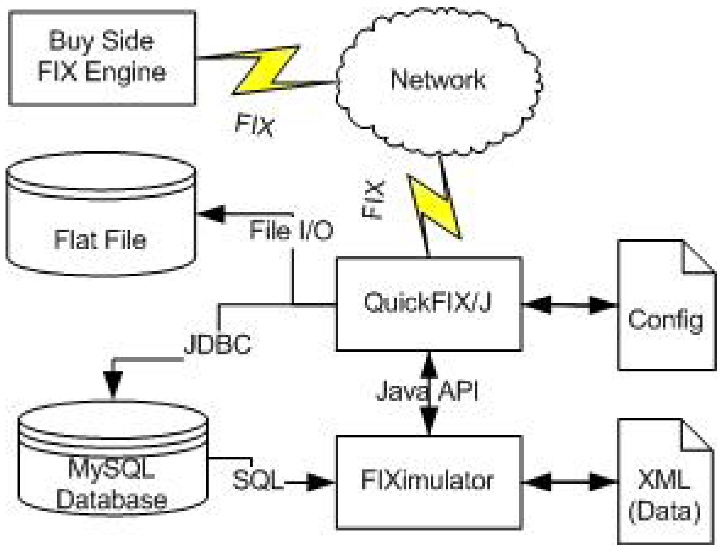


Figure 3 FIXimulator architecture

FIXimulator gives its users flexibility in controlling the FIX messages generated through the user interface. The user interface instructs QuickFIX/J to create and send the FIX messages through the network.

FIXimulator can be configured to log activities and messages to a flat file as well as a database as shown in Figure 3. The QuickFIX/J engine will write the messages to these as configured. FIXimulator can also access these data stores to retrieve the information to create reports of historical activity. FIXimulator uses flat files to store configurations in text format and also keeps a static data set of financial instruments in an XML format. Since FIXimulator only supports equity trading, the instrument record contains a number of stocks traded on US exchanges and their properties, such as identifiers and prices. These data are used to provide realistic values when running FIXimulator in automated mode.

In the example in Figure 3 FIXimulator is running in an automated mode generating IOIs and sending them to a buy-side client. The message below was generated while running in the automated mode:

```
8=FIX.4.2|9=197|35=6|34=4|49=FIXIMULATOR|52=20080720-
02:08:06.454|56=BANZAI|15=USD|22=5|23=I1216519686456|27=78400|28=N|44=4
2.0784|48=LAUR.OQ|54=1|55=LAUR|62=20080720-02:38:06.453|107=LAUREATE
EDUCATION INC|130=N|10=255|
```

The above FIXimulator generated message is an IOI for "Laureate Education Inc" at a price of $42.0784. The information for this security came from the following line of XML from the instruments.xml file located in the config directory.

```
   <instrument name="LAUREATE EDUCATION INC" ticker="LAUR"
sedol="2867689" ric="LAUR.OQ" cusip="518613104" price="47.07"/>
```

The FIX protocol supports the trading of many different financial products in addition to stocks or equities. It can also be used to trade bonds, foreign currencies, futures and other derivatives. Collectively these products can be referred to as financial instruments. Each instrument is stored as an element in this file with relevant information. In this case, we see that this instrument has a ticker symbol of "LAUR" which is populated in the Symbol (55) tag in the FIX message above. Tickers are assigned by the exchanges that a particular stock is traded on. The ticker symbols used by exchanges are just one of many different identifiers used by the industry. SEDOLs, RICs, and CUSIPs are examples of some others that are also used. SEDOL stands for "Stock Exchange Daily Official List" and is assigned by the London Stock Exchange, RIC is a "Reuters Instrument Code," and CUSIP stands for "Committee on Uniform Security Identification Procedures." All the above identifiers are widely used throughout the industry and are supported by FIXimulator.

In an IOI message there are two fields that are available to identify an instrument. In the above case we see that SecurityID (48) is set to "LAUR.OQ". That is a RIC as indicated by IDSource (22) being set to the value '5' as specified by the FIX specification.

Here is the configuration file used by FIXimulator at the time the above message was created:

```
[DEFAULT]
FIXimulatorLogToFile=Y
FIXimulatorAutoPendingCancel=N
FIXimulatorAutoPendingReplace=N
FIXimulatorLogToDB=Y
StartTime=00:00:00
JdbcUser=fiximulator
FIXimulatorSendOnBehalfOfCompID=N
FIXimulatorAutoAcknowledge=N
JdbcPassword=fiximulator
FIXimulatorAutoCancel=N
FIXimulatorAutoReplace=N
FIXimulatorPricePrecision=4
FIXimulatorSendOnBehalfOfSubID=N
SocketAcceptPort=9878
RefreshMessageStoreAtLogon=Y
BeginString=FIX.4.2
HeartBtInt=30
EndTime=00:00:00
JdbcDriver=com.mysql.jdbc.Driver
ConnectionType=acceptor
DataDictionary=FIX42.xml
FileStorePath=data
FIXimulatorCachedObjects=50
JdbcURL=jdbc:mysql://localhost:3306/quickfix
[SESSION]
OnBehalfOfSubID=DESK
FileLogPath=logs
TargetCompID=BANZAI
SenderCompID=FIXIMULATOR
OnBehalfOfCompID=BROKER
```

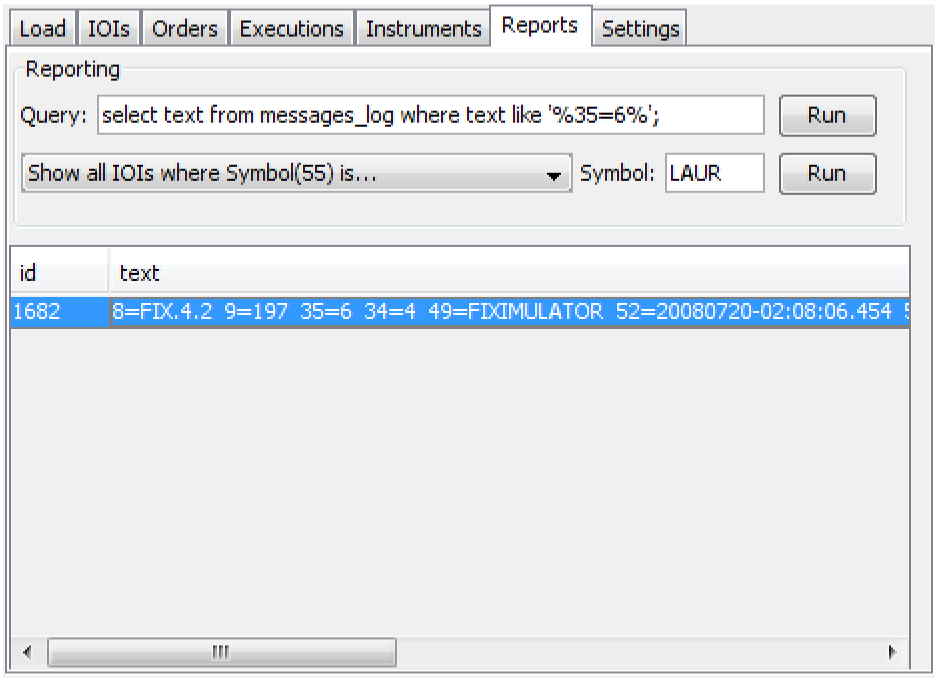
The Price (44) field in the above message was set to 42.0784 using 4 decimals for price precision. This setting is configured in the `FIXimulatorPricePrecision=4` parameter in the configuration file. Notice on Figure 3 that there is no direct link between the configuration file and FIXimulator. The configurations are encapsulated in a

Java object within QuickFIX/J and the settings are made available through the API. At runtime the above parameter is obtained by the getter method on the QuickFIX/J settings object.

```
pricePrecision = (int)settings.getLong("FIXimulatorPricePrecision");
```

Since settings are shared between QuickFIX/J and FIXimulator all settings used by FIXimulator are prefaced by "FIXimulator" in the configuration file.

Once the message is composed it is sent to the counterparty on the network and is persisted to a database and logged to a file. From the reporting interface of FIXimulator, the above message can be queried by using one of the prewritten SQL queries simply searching for IOIs where the Symbol (55) tag is "LAUR" as shown in Figure 4.



Query: select text from messages_log where text like '%35=6%';

Show all IOIs where Symbol(55) is... Symbol: LAUR

id	text
1682	8=FIX.4.2 9=197 35=6 34=4 49=FIXIMULATOR 52=20080720-02:08:06.454

Figure 4 SQL Query result

When this query is executed, FIXimulator runs the SQL query against the database and displays the resulting message. QuickFIX/J uses the JDBC connection to store the message to the database. FIXimulator can use SQL to query the database for the persisted messages.

Model-View-Controller Design Pattern

FIXimulator is designed using the classic Model-View-Controller (MVC) architecture and is broken down into packages that correspond to these layers. The MVC design pattern is widely used for applications with a Graphical User Interface (GUI). MVC design separates business logic, data access, and presentation layers of the application code (Gamma et al. 1995).

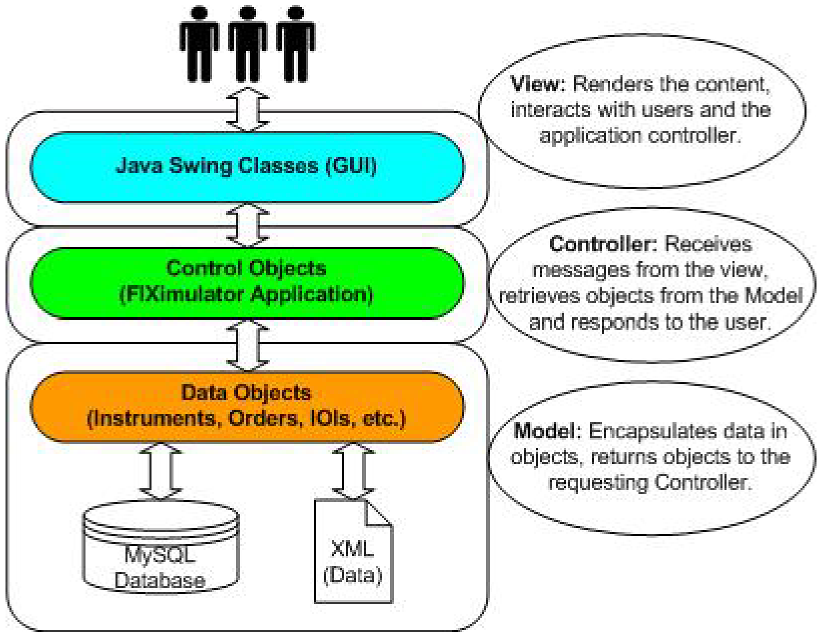

View: Renders the content, interacts with users and the application controller.
Java Swing Classes (GUI)
Control Objects (FIXimulator Application)
Controller: Receives messages from the view, retrieves objects from the Model and responds to the user.
Data Objects (Instruments, Orders, IOIs, etc.)
Model: Encapsulates data in objects, returns objects to the requesting Controller.
MySQL Database
XML (Data)

Figure 5 FIXimulator MVC implementation

Figure 5 shows how the MVC design pattern is implemented in FIXimulator. The following sections detail each component.

The Data Model

Orders, executions, IOIs, and instruments comprise the data (Model) used by FIXimulator. They are each represented by the individual Java classes presented in the core class diagram in Figure 6 with appropriate methods to access their properties. They are all part of the `edu.harvard.fas.zfeledy.fiximulator.core` application package.

As mentioned above, data about the financial instruments are stored in an XML file that is read into memory at runtime. Storing instruments in XML format also gives the user control over which instruments are loaded at runtime and how large the set of instruments may be. Each instrument is encapsulated in an object with appropriate methods to access their properties. These properties correspond to the attributes described in the XML file: name, ticker, SEDOL, RIC, CUSIP, and price.

Similarly, orders, IOIs, and executions are also encapsulated in Java objects as shown in Figure 6. FIXimulator can easily be enhanced to store these objects for future use in an XML file or a database, though this functionality is currently not supported.

Objects of these four object types, instruments, orders, IOIs, and executions are all collected into sets. This provides FIXimulator with the ability to reference the objects as needed. For example, every instrument is added to the InstrumentSet object which provides FIXimulator with the ability to obtain random instruments for IOIs or reference attributes of particular ones.

Similarly, IOIs, orders, and executions are also collected into sets. These sets are used to cache these objects in memory for performance reasons. All objects are cached as long as configured or needed. FIXimulator caches only the last 50 objects for display purposes by default and purges the rest from memory in order to limit memory consumption. This can be changed to cache the last 100 or 200 objects from the settings panel of the application at the expense of performance if needed.

Purging objects from memory that are no longer needed significantly enhances system performance. Should it be required to access these old objects then a persistence mechanism would need to be added.

The Application

All application code is in the `FIXimulatorApplication` class in the `edu.harvard.fas.zfeledy.fiximulator.core` package. This class is the controller of the MVC design pattern and it is one of the largest classes of FIXimulator. In the class diagram in Figure 6 it connects all the classes from the data model. All user controls available in the GUI ultimately correspond to methods in this class.

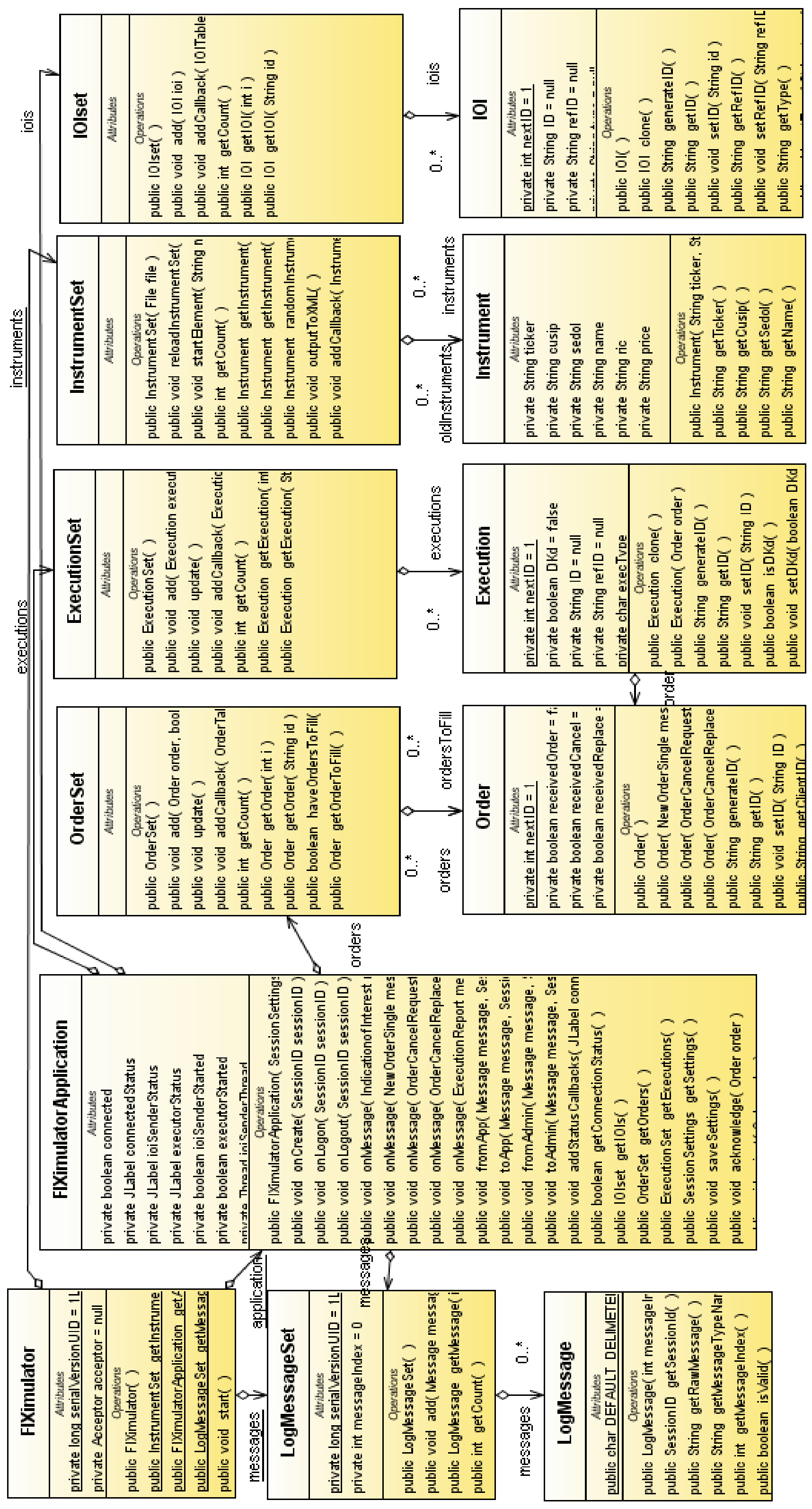


Figure 6 Core class diagram

The Graphical User Interface

A Java Swing GUI provides both the view to the data model and the user control of the business logic. The code for this functionality is separated out into the `edu.harvard.fas.zfeledy.fiximulator.ui` Java package. The user interface was designed using the Swing GUI Builder of NetBeans IDE 6.1 (NetBeans n. d.). The code generated by NetBeans uses the Java group layout for the layout manager. The group layout is a Java 6 feature (Sun Microsystems, 2006) and as a result, Java 6 is required to run FIXimulator.

The class diagram for the user interface is shown in Figure 7. Most trading applications utilize tables to display data to the user. FIXimulator is no exception and every set of data is displayed in tables. Each of these tables has corresponding models to display the data, some with renderer objects to render cells in color to distinguish between objects as needed. The reporting module uses the dynamic query table model to display tables based on the results of SQL queries. This table model allows the results to be displayed in a table fitting the result. All other user interface objects such as buttons, sliders, and menus are defined in the FIXimulatorFrame object.

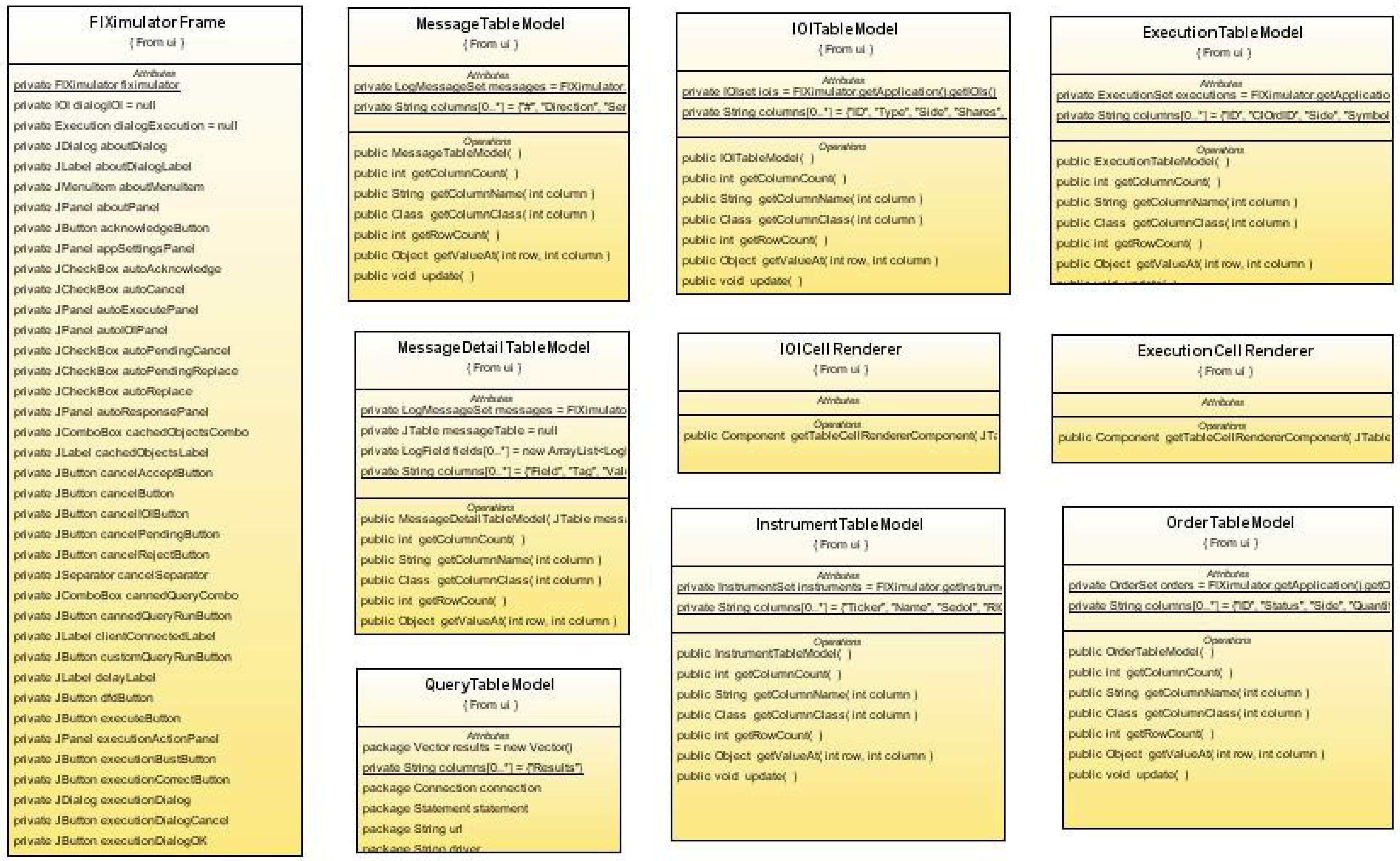

FIXimulatorFrame
(From ui)
Attributes
private FIXimulator fiximulator
private IOI dialogIOI = null
private Execution dialogExecution = null
private JDialog aboutDialog
private JLabel aboutDialogLabel
private JMenuItem aboutMenuItem
private JPanel aboutPanel
private JButton acknowledgeButton
private JPanel appSettingsPanel
private JCheckBox autoAcknowledge
private JCheckBox autoCancel
private JPanel autoExecutePanel
private JPanel autoIOIPanel
private JCheckBox autoPendingCancel
private JCheckBox autoPendingReplace
private JCheckBox autoReplace
private JPanel autoResponsePanel
private JComboBox cachedObjectsCombo
private JLabel cachedObjectsLabel
private JButton cancelAcceptButton
private JButton cancelButton
private JButton cancelIOIButton
private JButton cancelPendingButton
private JButton cancelRejectButton
private JSeparator cancelSeparator
private JComboBox cannedQueryCombo
private JButton cannedQueryRunButton
private JLabel clientConnectedLabel
private JButton customQueryRunButton
private JLabel delayLabel
private JButton dfdButton
private JButton executeButton
private JPanel executionActionPanel
private JButton executionBustButton
private JButton executionCorrectButton
private JDialog executionDialog
private JButton executionDialogCancel
private JButton executionDialogOK
MessageTableModel
(From ui)
Attributes
private LogMessageSet messages = FIXimulator.
private String columns[0..*] = {"#", "Direction", "Se
Operations
public MessageTableModel( )
public int getColumnCount( )
public String getColumnName(int column)
public Class getColumnClass(int column)
public int getRowCount( )
public Object getValueAt(int row, int column)
public void update( )
MessageDetailTableModel
(From ui)
Attributes
private LogMessageSet messages = FIXimulato
private JTable messageTable = null
private LogField fields[0..*] = new ArrayList<Log
private String columns[0..*] = {"Field", "Tag", "Val
Operations
public MessageDetailTableModel(JTable mess
public int getColumnCount( )
public String getColumnName(int column)
public Class getColumnClass(int column)
public int getRowCount( )
public Object getValueAt(int row, int column)
QueryTableModel
(From ui)
Attributes
package Vector results = new Vector()
private String columns[0..*] = {"Results"}
package Connection connection
package Statement statement
package String url
package String driver
IOITableModel
(From ui)
Attributes
private IOIset iois = FIXimulator.getApplication().getIOIs()
private String columns[0..*] = {"ID", "Type", "Side", "Shares",
Operations
public IOITableModel( )
public int getColumnCount( )
public String getColumnName(int column)
public Class getColumnClass(int column)
public int getRowCount( )
public Object getValueAt(int row, int column)
public void update( )
IOICellRenderer
(From ui)
Attributes
Operations
public Component getTableCellRendererComponent(JT
InstrumentTableModel
(From ui)
Attributes
private InstrumentSet instruments = FIXimulator.getInstrum
private String columns[0..*] = {"Ticker", "Name", "Sedol", "RI
Operations
public InstrumentTableModel( )
public int getColumnCount( )
public String getColumnName(int column)
public Class getColumnClass(int column)
public int getRowCount( )
public Object getValueAt(int row, int column)
public void update( )
ExecutionTableModel
(From ui)
Attributes
private ExecutionSet executions = FIXimulator.getApplicatio
private String columns[0..*] = {"ID", "ClOrdID", "Side", "Symbol
Operations
public ExecutionTableModel( )
public int getColumnCount( )
public String getColumnName(int column)
public Class getColumnClass(int column)
public int getRowCount( )
public Object getValueAt(int row, int column)
ExecutionCellRenderer
(From ui)
Attributes
Operations
public Component getTableCellRendererComponent(JTable
OrderTableModel
(From ui)
Attributes
private OrderSet orders = FIXimulator.getApplication().getO
private String columns[0..*] = {"ID", "Status", "Side", "Quanti
Operations
public OrderTableModel( )
public int getColumnCount( )
public String getColumnName(int column)
public Class getColumnClass(int column)
public int getRowCount( )
public Object getValueAt(int row, int column)
public void update( )

Figure 7 User interface class diagram

Utilities

FIXimulator displays the FIX messages within the application in a readable format as well as in a parsed format. To display the messages and the components in a readable format, certain manipulation is required by the application. FIXimulator classes replace the non-readable ASCII characters for display purposes and help with the parsing of the tag and value pairs to provide further details from the built-in QuickFIX/J data dictionary. To do this FIXimulator uses a module from another open source project, Log4FIX, developed by Brian Coyner. The package `edu.harvard.fas.zfeledy.fiximulator.util` contains helper classes from the Log4FIX project to assist with the parsing and displaying of messages in FIXimulator's messages and message detail viewers.

The QuickFIX/J API

In order to use the QuickFIX/J FIX engine an application has to implement the `Application` class defined by QuickFIX/J. FIXimulator's `FIXimulatorApplication` class implements this class and all the corresponding methods required for handling incoming and outgoing FIX messages. The application is then wrapped in either an initiator or an acceptor object. In the case of FIXimulator, it is wrapped as a `SocketAcceptor` along with all the settings including the logging configurations. The application starts when the start method is called on the `SocketAcceptor` object.

The application is responsible for handling all the business level logic. QuickFIX/J provides all the Java objects for the necessary FIX components to handle the FIX messaging. Every FIX message is a QuickFIX/J object with appropriate methods to access each of the FIX fields. Every FIX field is also a QuickFIX/J object to ensure that each field conforms to the FIX specification. Those objects have appropriate getter methods to obtain the values in a given field.

QuickFIX/J acts as a gatekeeper and allows only valid messages through. The application programmer need not be concerned about the validity of incoming messages. All invalid messages are rejected at the session level.

FIXimulator is responsible for the business level logic. FIXimulator maintains the business objects, and translates them to FIX messages at the time when they are ready to be sent. In one of the previous examples, FIXimulator would get a random instrument from the instrument set and generate an IOI. This IOI would then have to be turned into a FIX message. A new FIX field is created for each of the tags that need to be present. In the case of an IOI message the required fields are `IOIid`, `IOITransType`, `Side`, `IOIShares`, and `Symbol`. Each of these fields is a QuickFIX/J object so they simply need to be imported from the QuickFIX/J packages. All these fields need to be constructed before an actual IOI message can be instantiated. Once those fields are available, then an `IndicationofInterest` QuickFIX/J object can be instantiated. At this point, other conditionally required and optional fields may be added by using the appropriate setter methods. These methods ensure that only valid fields are being added to the FIX messages.

The application can implement a number of `onMessage` methods overloaded to handle each different incoming message. Again, all incoming FIX messages are received by these methods encapsulated in QuickFIX/J objects so the application programmer can extract the fields with the available getter methods. Each field is also encapsulated in a QuickFIX/J object with appropriate getter methods.

All business level logic can be implemented and translated to FIX by using the objects made available by the QuickFIX/J libraries.

The MySQL Database

FIXimulator can be configured to log to a database. The logging configurations are passed as an argument during the startup of the application and thus any changes require restarting the application. This is a limitation of the QuickFIX/J library since the logging configurations are passed as an argument when the application object is instantiated. Most trading applications are configured with a desired set of logging options that are never changed during runtime.

The database used by QuickFIX/J for logging is named `quickfix` and it contains 4 tables. QuickFIX/J uses the `messages` and `session` tables for maintaining session information. The `event_log` table logs all the connection and sequence number reset events. The `messages_log` is the primary table where all the FIX messages are logged.

The main benefit of logging to a database is that information can easily be extracted using simple SQL queries. A reporting feature within FIXimulator gives access to the data available in the database at runtime.

Configuration and Logging

The FIXimulator.cfg file contains all the configurations used by both QuickFIX/J and FIXimulator. It is a text based configuration file that is parsed by the QuickFIX/J application at runtime and read into a Java object. This object is then accessible through the API.

QuickFIX/J provides many options for logging. FIXimulator utilizes the console, flat file, and database logging options. The file and database options can both be configured within the FIXimulator configuration file and initiated at runtime. Changes to the logging options require a restart of the application.

The on screen or console logging is always turned on as it requires the least amount of resources. Even if the user should decide to turn off all logging through the settings to maximize performance, the console will always display all the events and the FIX messages.

Chapter 4 Installation

This chapter describes how to install FIXimulator along with a sample buy side application called Banzai. These two applications are delivered in zip archives named Banzai.zip and FIXimulator.zip. When the archives are extracted, directories with the same names are created. FIXimulator requires Java SE Runtime Environment (JRE) 6 which can be downloaded from http://java.sun.com/javase/downloads/index.jsp.

To launch FIXimulator, the fiximulator.bat file needs to be executed. FIXimulator will wait for a connection. Similarly, to launch Banzai, the banzai.bat file needs to be run. Banzai is a sample buy side application that initiates a connection to FIXimulator when the program starts. Both applications are configured by default to connect to each other.

FIXimulator has an optional functionality to log its messages to a MySQL database and to create reports from the database. Configuring a MySQL database is beyond the scope of this document but downloads and installation instructions are available from the MySQL website, http://www.mysql.com/. If a MySQL database is running on the same machine as FIXimulator then a new database and a user account will need to be created. To create these simply execute the create.bat in the FIXimulator\mysql directory by double clicking the file. The file contains the following instructions.

```
mysql -u root -p --execute="source mysql.sql";
pause;
```

This script calls the mysql.sql script to be executed on the database as the root user.

```
source quickfix_database.sql;
source fiximulator_user;
source sessions_table.sql;
source messages_table.sql;
source messages_log_table.sql;
source event_log_table.sql;
```

The first script called in this file simply creates a database “quickfix”.

```
DROP DATABASE IF EXISTS quickfix;
CREATE DATABASE quickfix;
```

The fiximulator_user.sql script creates a “fiximulator” user in the database. This user is granted all privileges to the quickfix database.

```
grant all privileges on quickfix.*
to 'fiximulator'@'localhost'
identified by 'fiximulator';
```

The script then calls the sessions_table.sql script to create the sessions table.

```
USE quickfix;

DROP TABLE IF EXISTS sessions;

CREATE TABLE sessions (
  beginstring CHAR(8) NOT NULL,
  sendercompid VARCHAR(64) NOT NULL,
  sendersubid VARCHAR(64) NOT NULL,
  senderlocid VARCHAR(64) NOT NULL,
  targetcompid VARCHAR(64) NOT NULL,
  targetsubid VARCHAR(64) NOT NULL,
  targetlocid VARCHAR(64) NOT NULL,
  session_qualifier VARCHAR(64) NOT NULL,
  creation_time DATETIME NOT NULL,
  incoming_seqnum INT NOT NULL,
  outgoing_seqnum INT NOT NULL,
  PRIMARY KEY (beginstring, sendercompid, sendersubid, senderlocid,
                        targetcompid, targetsubid, targetlocid,
session_qualifier)
);
```

Similarly, the next scripts that are called, messages_table.sql, messages_log_table.sql, event_log_table.sql create the other tables.

Once the mysql.sql file runs to success a new database “quickfix” is created along with a user “fiximulator” and 4 tables “sessions”, “event_log”, “messages”, and “messages_log” that are used by FIXimulator. This allows FIXimulator to log to a database and the option can be enabled in the FIXimulator settings.

Chapter 5 User Guide

This chapter describes the functionalities of FIXimulator and the actions that are performed by the Banzai application.

Configuration

FIXimulator is configured by default to communicate with the customized version of Banzai that comes with the application so no additional configuration is required when using these two applications together. This configuration is shown in Figure 8.

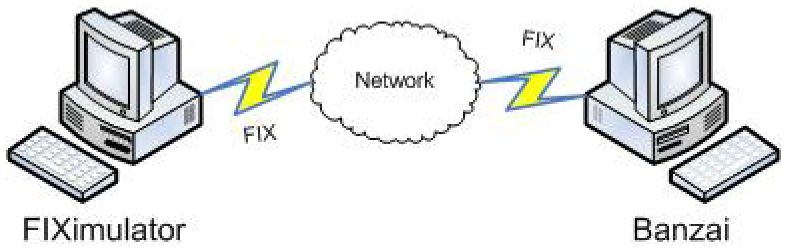


Figure 8 Default FIXimulator and Banzai setup

All application level configurations are accessible from the graphical user interface. All settings are stored in a file called FIXimulator.cfg in the config directory. If FIXimulator is connected to another application then the following parameters must be adjusted as needed in the FIXimulator configuration file:

```
SocketAcceptPort=9878
TargetCompID=BANZAI
SenderCompID=FIXIMULATOR
```

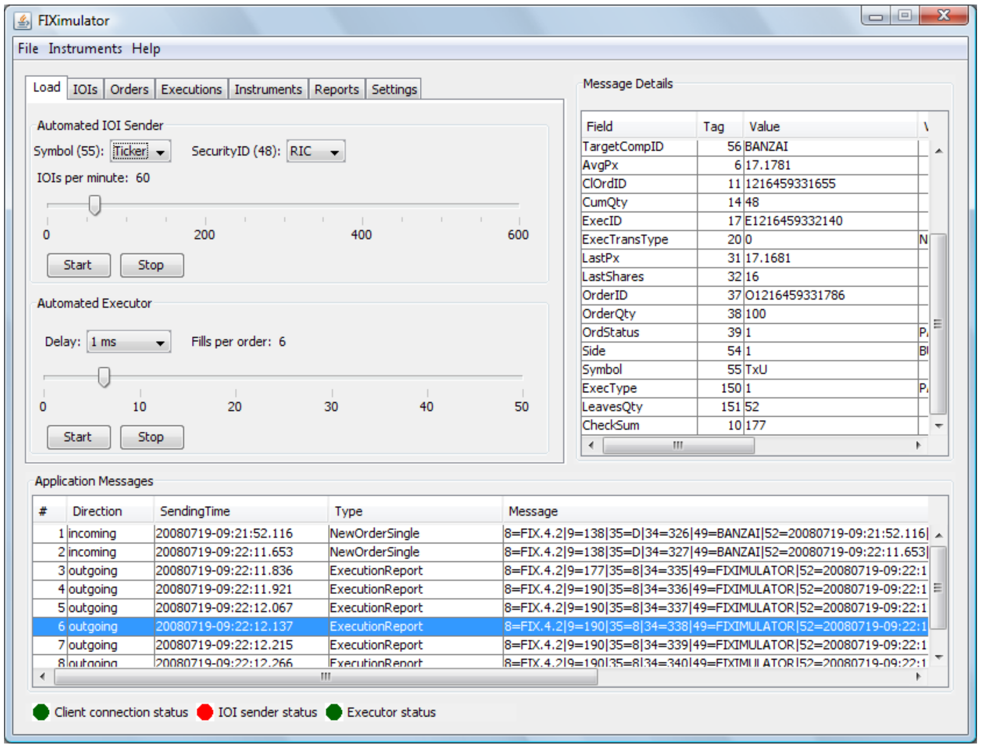
These parameters will tell FIXimulator what port to expect a connection on and what session level identifiers are needed to establish a valid FIX connection.

Starting the Application

To start FIXimulator the fiximulator.bat file needs to be run by double clicking the file. This will open a standard console where the application outputs the various activities as well as a Swing user interface. FIXimulator will wait to receive a connection from a buy-side application. If Banzai is used as the buy-side application, execute the banzai.bat file and a connection will be made between the applications automatically.

FIXimulator Overview

The figure below is a screenshot of FIXimulator's user interface.



Symbol (55): Ticker SecurityID (48): RIC

IOIs per minute: 60

Delay: 1 ms Fills per order: 6

Field	Tag	Value	
TargetCompID	56	BANZAI	
AvgPx	6	17.1781	
ClOrdID	11	1216459331655	
CumQty	14	48	
ExecID	17	E1216459332140	
ExecTransType	20	0	N
LastPx	31	17.1681	
LastShares	32	16	
OrderID	37	O1216459331786	
OrderQty	38	100	
OrdStatus	39	1	P
Side	54	1	B
Symbol	55	TxU	
ExecType	150	1	P
LeavesQty	151	52	
CheckSum	10	177	

#	Direction	SendingTime	Type	Message
1	incoming	20080719-09:21:52.116	NewOrderSingle	8=FIX.4.2|9=138|35=D|34=326|49=BANZAI|52=20080719-09:21:52.116|
2	incoming	20080719-09:22:11.653	NewOrderSingle	8=FIX.4.2|9=138|35=D|34=327|49=BANZAI|52=20080719-09:22:11.653|
3	outgoing	20080719-09:22:11.836	ExecutionReport	8=FIX.4.2|9=177|35=8|34=335|49=FIXIMULATOR|52=20080719-09:22:1
4	outgoing	20080719-09:22:11.921	ExecutionReport	8=FIX.4.2|9=190|35=8|34=336|49=FIXIMULATOR|52=20080719-09:22:1
5	outgoing	20080719-09:22:12.067	ExecutionReport	8=FIX.4.2|9=190|35=8|34=337|49=FIXIMULATOR|52=20080719-09:22:1
6	outgoing	20080719-09:22:12.137	ExecutionReport	8=FIX.4.2|9=190|35=8|34=338|49=FIXIMULATOR|52=20080719-09:22:1
7	outgoing	20080719-09:22:12.215	ExecutionReport	8=FIX.4.2|9=190|35=8|34=339|49=FIXIMULATOR|52=20080719-09:22:1
8	outgoing	20080719-09:22:12.266	ExecutionReport	8=FIX.4.2|9=190|35=8|34=340|49=FIXIMULATOR|52=20080719-09:22:1

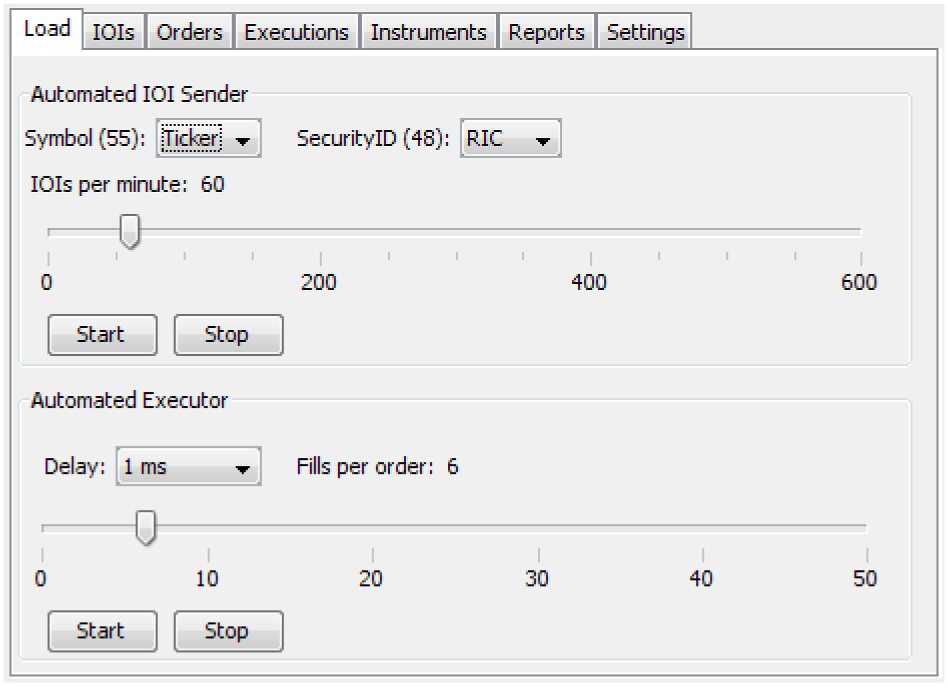
● Client connection status ● IOI sender status ● Executor status

Figure 9 FIXimulator user interface

The user interface has three main parts. The main panel on the top left is where the user controls are located for the various modes of operation. The bottom of the screen contains a table of the application level FIX messages. A raw FIX message is difficult to read so the application message panel is linked to the message details panel in the upper right hand side which parses selected messages into a readable format. At the very bottom of the interface is a status bar that displays the connection status as well as the statuses of the automated IOI Sender and the Automated Executor threads.

The Main Panel

The main panel in the following figure is where the user controls FIXimulator. This panel is a pane made up of seven tabs organized to handle the different functionalities.


Load | IOIs | Orders | Executions | Instruments | Reports | Settings

Automated IOI Sender

Symbol (55): Ticker   SecurityID (48): RIC

IOIs per minute: 60

0 200 400 600

Start Stop

Automated Executor

Delay: 1 ms   Fills per order: 6

0 10 20 30 40 50

Start Stop

Figure 10 Main panel

The Load Tab

The Load tab displayed in Figure 10 gives the user control over the simplest forms of operation. If the client is connected, indicated by a green dot for the connection status in the status bar then IOIs can automatically be sent using the default set of instruments that are loaded when FIXimulator starts. Pressing the Start button in the Automated IOI Sender section of the Load tab will begin sending IOIs to the client at the rate indicated above the slider control. The rate can be adjusted by moving the slider to the desired rate. When FIXimulator starts the rate is set to one IOI per second. If IOIs are no longer desired, pressing the stop button will stop the IOI Sender thread. The indicator in the status bar will turn red showing that the IOI Sender thread has stopped.

IOI messages have two fields for identifying the instrument that is being advertised. These fields are Symbol (55) and SecurityID (48). To test how receiving applications process identifying instruments these tags can be adjusted to the various identifiers available in the instrument record.

The bottom section of the Load panel is used to control the Automated Executor. When it is started, it will automatically process orders that are sent to FIXimulator. This workflow is detailed in Figure 11 below.

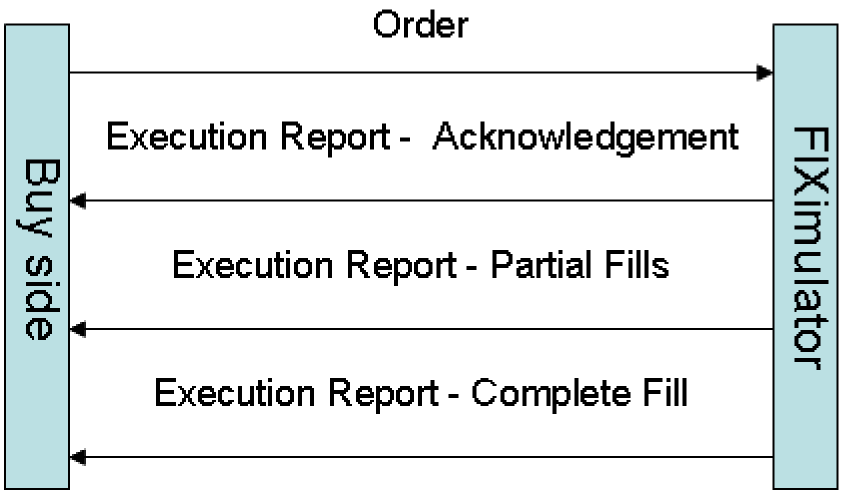
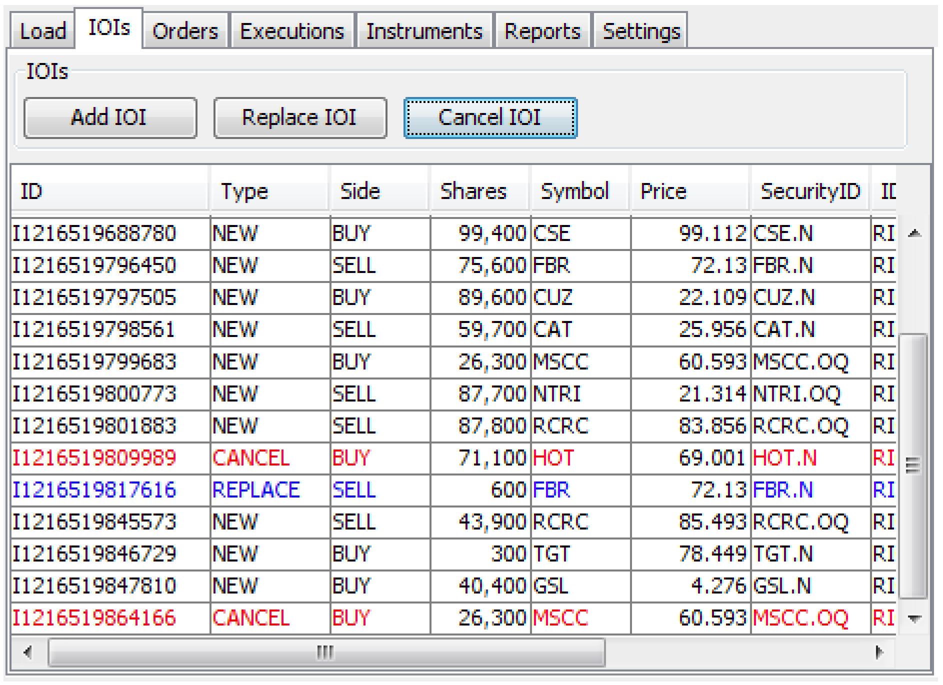

Order
Execution Report - Acknowledgement
Execution Report - Partial Fills
Execution Report - Complete Fill
Buy side
FIXimulator

Figure 11 Automated Executor workflow

The above two modes of operations could be run simultaneously thereby placing receiving buy-side systems under load. Running both threads at the same time generating random IOIs and automatically filling orders FIXimulator would send 3,000 application messages per minute on a 1.8 GHz AMD Sempron, and about 7,000 on a 2.4 GHz Intel Duo Core running on a localhost with Banzai.

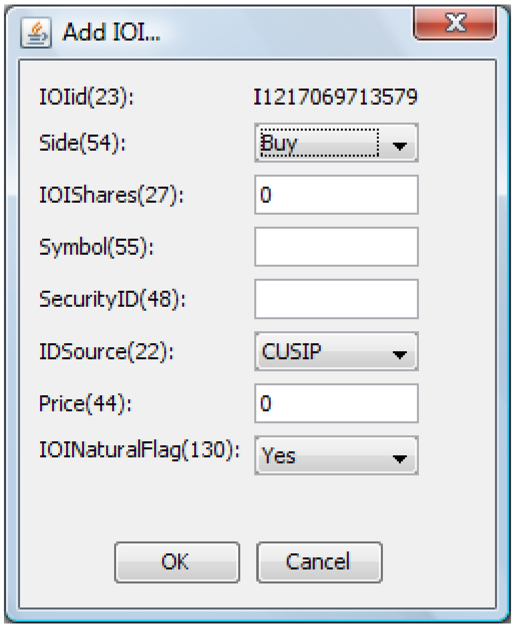
The IOI Tab



ID	Type	Side	Shares	Symbol	Price	SecurityID	IE
I1216519688780	NEW	BUY	99,400	CSE	99.112	CSE.N	RI
I1216519796450	NEW	SELL	75,600	FBR	72.13	FBR.N	RI
I1216519797505	NEW	BUY	89,600	CUZ	22.109	CUZ.N	RI
I1216519798561	NEW	SELL	59,700	CAT	25.956	CAT.N	RI
I1216519799683	NEW	BUY	26,300	MSCC	60.593	MSCC.OQ	RI
I1216519800773	NEW	SELL	87,700	NTRI	21.314	NTRI.OQ	RI
I1216519801883	NEW	SELL	87,800	RCRC	83.856	RCRC.OQ	RI
I1216519809989	CANCEL	BUY	71,100	HOT	69.001	HOT.N	RI
I1216519817616	REPLACE	SELL	600	FBR	72.13	FBR.N	RI
I1216519845573	NEW	SELL	43,900	RCRC	85.493	RCRC.OQ	RI
I1216519846729	NEW	BUY	300	TGT	78.449	TGT.N	RI
I1216519847810	NEW	BUY	40,400	GSL	4.276	GSL.N	RI
I1216519864166	CANCEL	BUY	26,300	MSCC	60.593	MSCC.OQ	RI

Figure 12 IOI panel

The IOI tab displayed in Figure 12 gives the user control over individual IOIs. By default, the table displays the details of the last 50 IOIs that have been sent. From this panel the user is able to add, replace, and cancel IOI messages. When the “Add IOI” button is pressed a dialog is opened giving the user control over the details of the IOI message to be generated. The dialog in Figure 13 gives the user control over the key fields of an IOI message. When the required fields are entered and the user clicks the OK button the message is sent to the client.



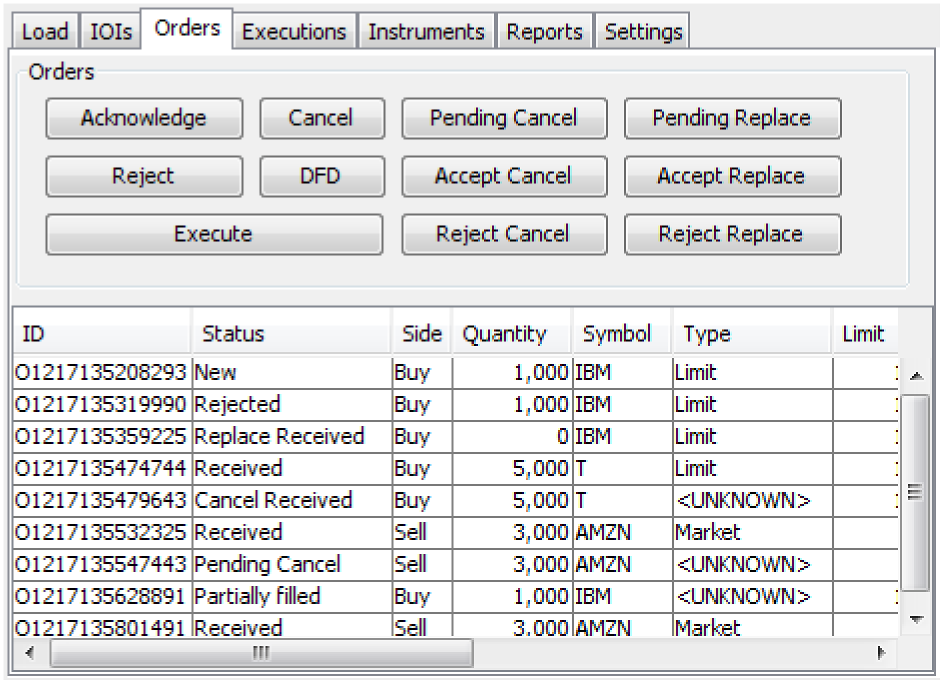
IOIid(23):	I1217069713579
Side(54):	Buy
IOIShares(27):	0
Symbol(55):	
SecurityID(48):	
IDSource(22):	CUSIP
Price(44):	0
IOINaturalFlag(130):	Yes

Figure 13 IOI dialog

Once IOIs are sent, they appear in the table shown in Figure 12. Selecting an IOI from the table allows the user to cancel or replace it. Pressing “Cancel IOI” with an IOI selected immediately sends a cancel message. The “Replace IOI” button displays the IOI dialog populated by the values of the selected IOI. The user can change any of the attributes and pressing “OK” will send a correction message to the client with the new details.

The Orders Tab

The “Orders” tab shown in Figure 14 is the most complicated one as it has to be able to report the statuses of the orders received by FIXimulator. Once an order is received by the application it will be displayed in the “Orders” panel and a number of workflows may follow. Actions can be done by selecting the order in the orders table and pressing the appropriate button. The order can be acknowledged by the sell side or simply rejected. If an order is rejected then that workflow is over. If however the order is acknowledged or accepted then several options are again available. Part or all of the order can be executed. In order to execute an order, the "Execute" button needs to be pressed with the order selected in the table. This will display a dialog asking the user to enter the number of shares and the price at which the order should be executed. For the purposes of this guide, let's assume that the order is partially executed. At this point, the sell side will either cancel the remaining part of the order or send a DFD (Done for the Day) message notifying the buy side that no further executions will follow on this order. Clicking the "Cancel" or the "DFD" buttons will send these messages.



ID	Status	Side	Quantity	Symbol	Type	Limit
O1217135208293	New	Buy	1,000	IBM	Limit	
O1217135319990	Rejected	Buy	1,000	IBM	Limit	
O1217135359225	Replace Received	Buy	0	IBM	Limit	
O1217135474744	Received	Buy	5,000	T	Limit	
O1217135479643	Cancel Received	Buy	5,000	T	<UNKNOWN>	
O1217135532325	Received	Sell	3,000	AMZN	Market	
O1217135547443	Pending Cancel	Sell	3,000	AMZN	<UNKNOWN>	
O1217135628891	Partially filled	Buy	1,000	IBM	<UNKNOWN>	
O1217135801491	Received	Sell	3,000	AMZN	Market	

Figure 14 Screenshot of the Orders tab

The sell side generated all the above actions on the orders so far without any changes having been requested by the buy side once the order was received. However, during the life of the order the buy side may decide to cancel or change an order currently being worked by the sell side. The buttons on the right side of the panel in Figure 14 simulate the actions of the sell side in these cases.

While an order is being worked, the buy side may decide to cancel or replace the order. When a request like this is made, the sell side usually sends a “pending” message acknowledging the receipt of the cancel or replace request to advise the buy side that they are evaluating whether the request can be honored. At times, the requests cannot be accommodated. Filled orders cannot be cancelled and orders with partial fills cannot change limits outside of the average price of the existing fills.

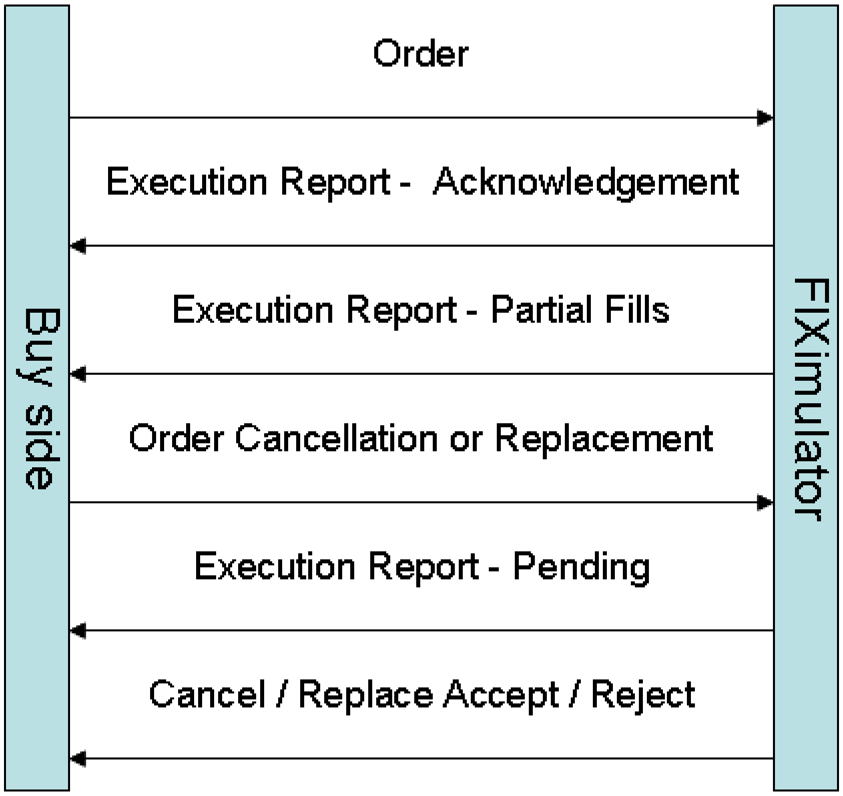
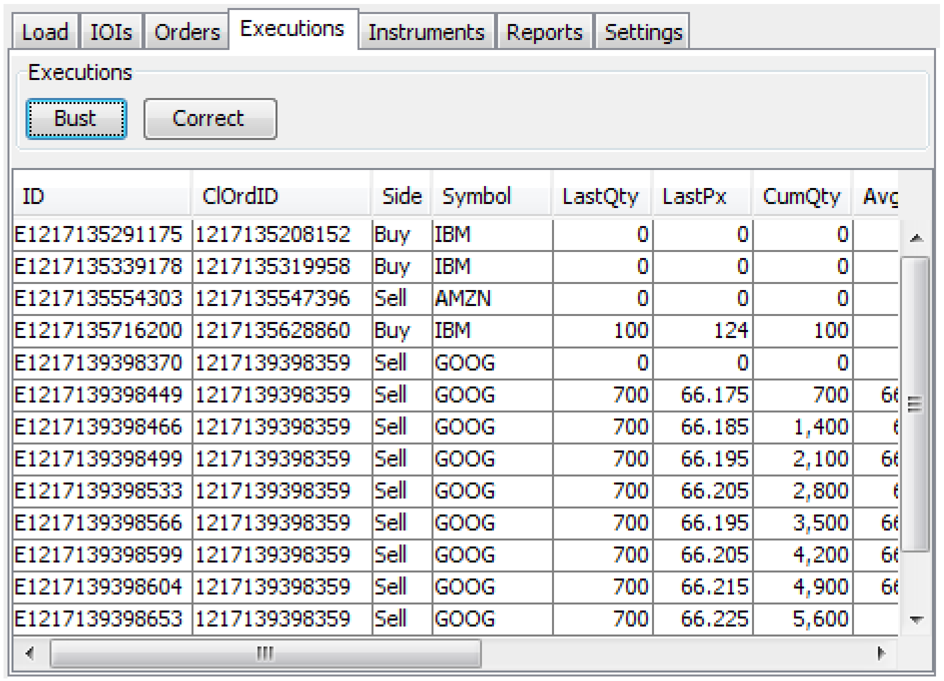

Buy side
FIXimulator
Order
Execution Report - Acknowledgement
Execution Report - Partial Fills
Order Cancellation or Replacement
Execution Report - Pending
Cancel / Replace Accept / Reject

Figure 15 Order cancel / replace workflow

The six buttons on the right side of the orders panel in Figure 14 support all the state changes in Figure 15. The “Status” column in the order table displays the various statuses throughout these state changes. When a cancel or replace message is received it is indicated in this column. To acknowledge the receipt and send pending messages, the order is selected and the appropriate button is pressed. Pressing the “Accept” or “Reject” buttons corresponding to the request will send a message to the buy side advising them of the status of the order.

The Executions Tab

The executions tab displays all the executions that are sent by FIXimulator. Executions are generated either by the automated executor or manually from the orders panel. This panel simply displays the executions and gives the user control over cancelling, generally referred to as busting, or correcting the executions.



ID	ClOrdID	Side	Symbol	LastQty	LastPx	CumQty	Avg
E1217135291175	1217135208152	Buy	IBM	0	0	0	
E1217135339178	1217135319958	Buy	IBM	0	0	0	
E1217135554303	1217135547396	Sell	AMZN	0	0	0	
E1217135716200	1217135628860	Buy	IBM	100	124	100	
E1217139398370	1217139398359	Sell	GOOG	0	0	0	
E1217139398449	1217139398359	Sell	GOOG	700	66.175	700	66
E1217139398466	1217139398359	Sell	GOOG	700	66.185	1,400	6
E1217139398499	1217139398359	Sell	GOOG	700	66.195	2,100	66
E1217139398533	1217139398359	Sell	GOOG	700	66.205	2,800	6
E1217139398566	1217139398359	Sell	GOOG	700	66.195	3,500	66
E1217139398599	1217139398359	Sell	GOOG	700	66.205	4,200	66
E1217139398604	1217139398359	Sell	GOOG	700	66.215	4,900	66
E1217139398653	1217139398359	Sell	GOOG	700	66.225	5,600	

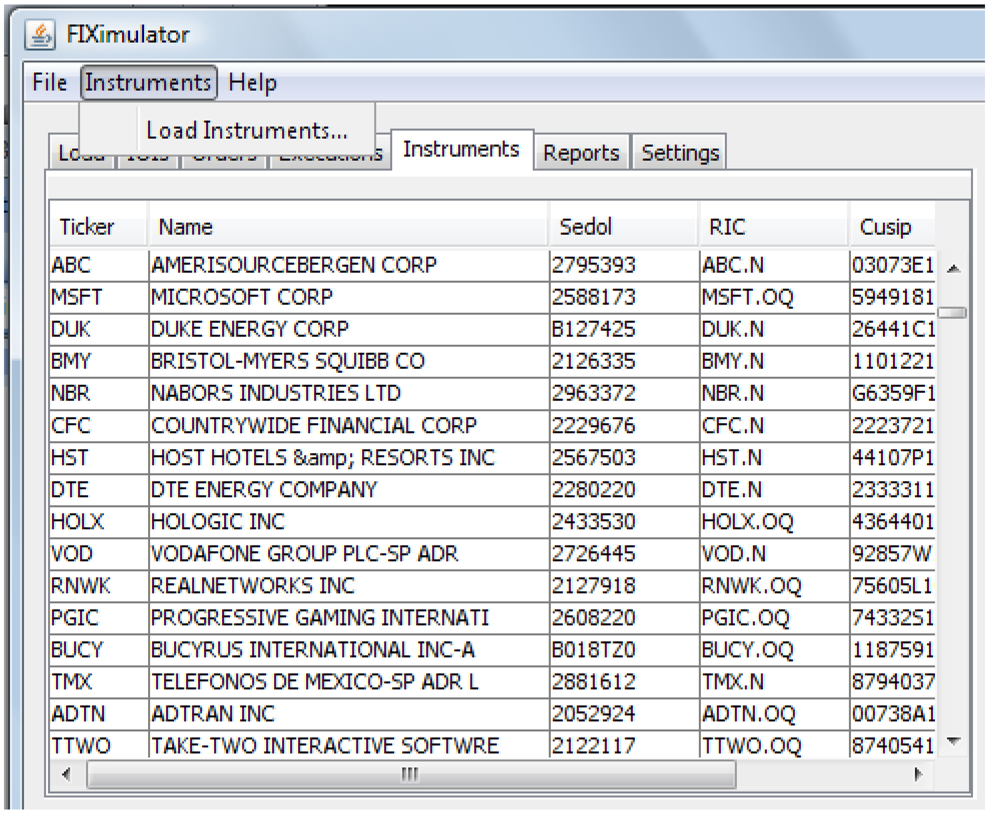
Figure 16 Executions Panel

Figure 16 shows the interface for busting and correcting executions. To bust an execution, the “Bust” button needs to be clicked with an execution selected from the table. This will send a report to the buy side advising them of the change.

When an execution is selected from the table and the “Correct” button is pressed, the execution dialog will appear populated by the shares and the price of the selected execution. Change the desired attribute and press “OK” to send a correction message to the counterparty.

The Instruments Tab

The instrument tab displayed in Figure 17 displays the current instrument set. FIXimulator uses this instrument set to provide accurate security identifiers and reasonable prices when running in an automated mode.



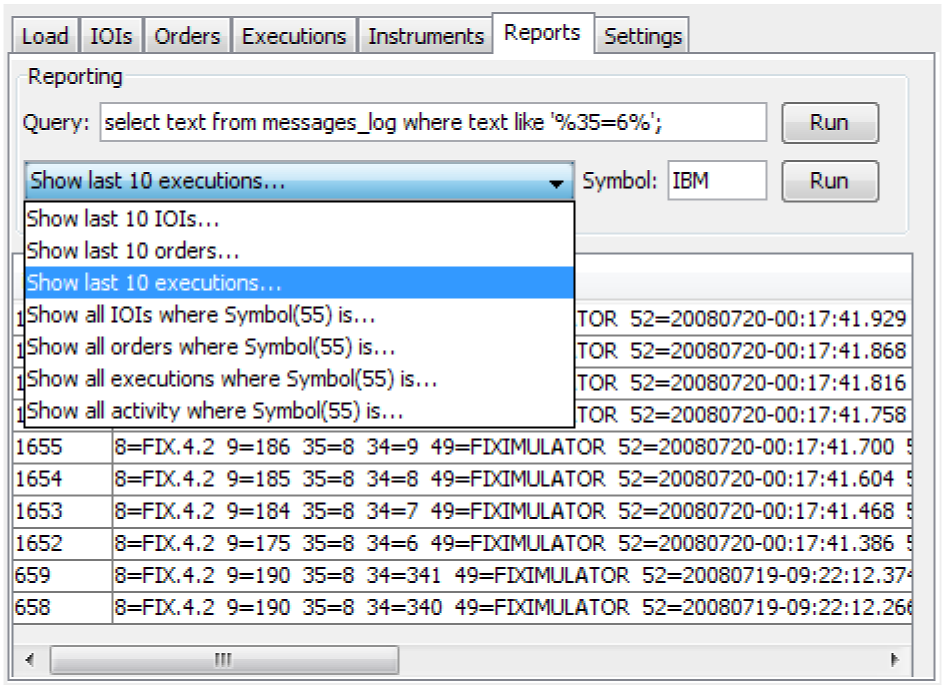
Ticker	Name	Sedol	RIC	Cusip
ABC	AMERISOURCEBERGEN CORP	2795393	ABC.N	03073E1
MSFT	MICROSOFT CORP	2588173	MSFT.OQ	5949181
DUK	DUKE ENERGY CORP	B127425	DUK.N	26441C1
BMY	BRISTOL-MYERS SQUIBB CO	2126335	BMY.N	1101221
NBR	NABORS INDUSTRIES LTD	2963372	NBR.N	G6359F1
CFC	COUNTRYWIDE FINANCIAL CORP	2229676	CFC.N	2223721
HST	HOST HOTELS &amp; RESORTS INC	2567503	HST.N	44107P1
DTE	DTE ENERGY COMPANY	2280220	DTE.N	2333311
HOLX	HOLOGIC INC	2433530	HOLX.OQ	4364401
VOD	VODAFONE GROUP PLC-SP ADR	2726445	VOD.N	92857W
RNWK	REALNETWORKS INC	2127918	RNWK.OQ	75605L1
PGIC	PROGRESSIVE GAMING INTERNATI	2608220	PGIC.OQ	7433251
BUCY	BUCYRUS INTERNATIONAL INC-A	B018TZ0	BUCY.OQ	1187591
TMX	TELEFONOS DE MEXICO-SP ADR L	2881612	TMX.N	8794037
ADTN	ADTRAN INC	2052924	ADTN.OQ	00738A1
TTWO	TAKE-TWO INTERACTIVE SOFTWRE	2122117	TTWO.OQ	8740541

Figure 17 Instrument panel

The instruments are stored in an XML file in the config directory. There are two files in this directory to demonstrate switching instrument sets. The default that loads at runtime is `instruments.xml`. Loading the `instruments_small.xml` file will provide the user with a smaller set of instruments. To switch between instrument sets go to the "Instruments" item in the menu bar and select "Load Instruments…" This will open a dialog to select and load a new file. If a valid new file is selected then the new instrument set will immediately be available otherwise the old set will be reset. This can be used to create a smaller set of instruments to hit working orders on a buy side application. Also, the user may desire an international set of instruments as opposed to the default set, which includes only US instruments. There are many other creative ways to utilize this functionality of loading custom instrument sets as well.

The Reports Tab

The "Reports" tab is an interface to the MySQL database if one is configured with FIXimulator. Figure 18 shows a screenshot of the interface from a configuration where FIXimulator is connected to a database.



Query: select text from messages_log where text like '%35=6%';

Show last 10 executions... Symbol: IBM

```
Show last 10 IOIs...
Show last 10 orders...
Show last 10 executions...
Show all IOIs where Symbol(55) is...
Show all orders where Symbol(55) is...
Show all executions where Symbol(55) is...
Show all activity where Symbol(55) is...
```

```
1                                                    TOR  52=20080720-00:17:41.929
1                                                    TOR  52=20080720-00:17:41.868
1                                                    TOR  52=20080720-00:17:41.816
1                                                    TOR  52=20080720-00:17:41.758
1655   8=FIX.4.2  9=186  35=8  34=9  49=FIXIMULATOR  52=20080720-00:17:41.700
1654   8=FIX.4.2  9=185  35=8  34=8  49=FIXIMULATOR  52=20080720-00:17:41.604
1653   8=FIX.4.2  9=184  35=8  34=7  49=FIXIMULATOR  52=20080720-00:17:41.468
1652   8=FIX.4.2  9=175  35=8  34=6  49=FIXIMULATOR  52=20080720-00:17:41.386
659    8=FIX.4.2  9=190  35=8  34=341  49=FIXIMULATOR  52=20080719-09:22:12.37
658    8=FIX.4.2  9=190  35=8  34=340  49=FIXIMULATOR  52=20080719-09:22:12.26
```

Figure 18 Reports panel

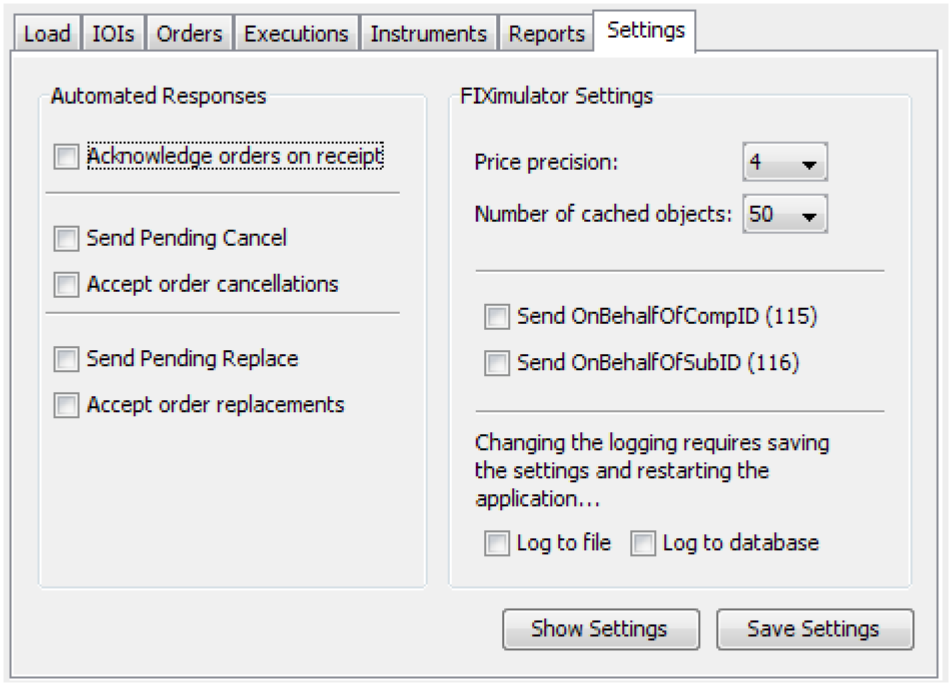
FIX messages contain all information about each message involved in a trading transaction. This fact, combined with the power of SQL allows the user to extract information about what happened in a particular workflow. FIXimulator allows the user to run queries directly against the database to which the FIX engine is logging messages. Familiarity with the messages and the database tables gives the user great flexibility over what information is extracted. When FIXimulator is first started the query text field is populated with a simple query to look for messages with "35=6" indicating that only IOIs should be returned. Running this query will return all the IOIs stored in the database. The user may also write any custom queries and execute them by pressing the "Run" button.

In addition to custom queries, several sample reports are also available. These can be accessed from the dropdown below the custom queries. Some of these queries utilize the Symbol (55) field of a message that indicates the instrument and is required on most application message types.

The queries return the dataset into a multiple selection table where the user may decide to copy the results into another application for further analysis.

The Settings Tab

All of FIXimulator's settings are stored in a text file named FIXimulator.cfg located in the config directory. When the application is started all the settings are pulled into a Java object and they are accessible to the application during runtime. All settings that may affect the operation of FIXimulator can be adjusted during runtime through the "Settings" tab shown in Figure 19.



Automated Responses

☐ Acknowledge orders on receipt

☐ Send Pending Cancel
☐ Accept order cancellations

☐ Send Pending Replace
☐ Accept order replacements

FIXimulator Settings

Price precision: 4
Number of cached objects: 50

☐ Send OnBehalfOfCompID (115)
☐ Send OnBehalfOfSubID (116)

Changing the logging requires saving the settings and restarting the application...

☐ Log to file ☐ Log to database

Figure 19 Settings tab

Should the user desire to automatically acknowledge orders on receipt, or send pending messages when cancel and replace requests are received, the appropriate tick boxes can be selected and the runtime settings are immediately updated. The user may also want to store a varying number of objects in memory at runtime. All objects that are generated during the operation of FIXimulator are purged for performance reasons. By default, only the last 50 orders, executions, and FIX messages are stored. The user can change the number of available objects by changing the value in the combo box labeled "Number of cached objects." The largest set is limited to 200 beyond which the user should elect to log to a database and extract any number of objects desired. Similarly, the user can specify the default price precision to be used when calculating average prices on messages where this is relevant.

There are many routing configurations for FIX communications. The default settings for FIXimulator are configured to simulate a point-to-point model where the communication takes place directly between two counterparties. In this configuration, each party has an ID and they get populated into the SenderCompID (49) and TargetCompID (56) fields. In the default configuration where Banzai is talking to FIXimulator, the fields would be populated as 49=BANZAI and 56=FIXIMULATOR when Banzai sends a message to FIXimulator and 49= FIXIMULATOR and 56= BANZAI when FIXimulator is sending the message.

In actual practice there are many other configurations for routing messages. A number of businesses specialize in creating FIX hubs and connecting many people from the different sides. There are many advantages to electing this model. With a single relationship, a buy side can be connected to many brokers or a broker can be connected to many buy sides. In these configurations, buy sides would always be targeting their FIX hubs asking them to forward the message to the party specified in the DeliverToCompID

(128) field. When responses are received the hubs populate the responses with the OnBehalfOfCompID (115). Since FIXimulator is a sell side application, if this third party configuration is to be followed then it would have to send messages with the OnBehalfOfCompID (115) field populated. The value to send can be configured in the FIXimulator.cfg file and ticking the box to send this value in the settings panel will add it to the FIX messages thereby giving FIXimulator the ability to simulate transactions involving a third party.

There is one last routing configuration that is used to send messages to the appropriate department at the receiving party. This routing configuration may involve different desks at a broker. When messages are returned they are populated in the OnBehalfOfSubID (116) field which works similarly to the above.

Most FIXimulator settings can be adjusted during runtime and take immediate effect. The “Show Settings” button will display the current settings in memory to the console, including the ones that cannot be adjusted during runtime. The “Save Settings” button will save the current settings for the next time FIXimulator runs by overwriting the FIXimulator.cfg file with the current settings in memory.

FIXimulator can also be configured to log its messages to a flat file or a database in addition to the console. The default is to log only to a screen since most receiving applications do their own logging and this allows for the highest performance when running in a load mode. Ticking the appropriate check boxes, saving the settings, and restarting the application will log to a file, database, or both. Again, this is due to a limitation within the QuickFIX/J API which would require significant work to change.

Chapter 6 Summary and Conclusions

Though many commercial applications are available to generate application level FIX messages, only a few are open source. As this project developed the ideas were presented to the open source community and it generated great interest. Developers from the QuickFIX/J project and the open source buy side OMS marketcetera project both contacted me with words of encouragement looking forward to the availability of FIXimulator in the public space.

One of the biggest obstacles, as in most FIX applications, was the flexibility of the FIX protocol. Flexible technologies are often difficult to implement as the flexibility often gives way to many ambiguities. For example, when an order is sent, a response is expected to either acknowledge or reject the order. How should the sending application treat the order if a response is not received and the order is to be cancelled? Historically, no cancel message was sent and further responses were simply rejected. In today's fast moving, low latency markets it is now preferred by many applications to send a cancel message even though it may seem out of turn. To overcome these types of obstacles I relied on my personal experience and familiarity in the equities space where the protocol has been in use for well over a decade.

There are many ways to expand FIXimulator. Adding support for other versions of the protocol would probably be the most important as there are slight differences among the versions that are being used in the industry. Parsing the FIX messages returned from the database would significantly enhance the reporting functionality. Not requiring the user to have familiarity with SQL would also make reporting more user-friendly.

One of the suggestions from the QuickFIX/J project was to completely separate out the GUI code so that FIXimulator could run entirely in a server mode. Implementing this would likely result in higher performance. This will likely be one of the first pieces developed once the project moves to the public space.

QuickFIX/J also offers a scripting language for the actual scripting of FIX messages to be used as test cases. Adding an intuitive interface for this functionality would provide great flexibility to the functional testing features of the application.

Building FIXimulator was a very rewarding experience that thought me a great deal about software design. I am looking forward to seeing how the application will grow over time in the public space and the directions it may take based on public demand.

Glossary

Buy side
The side of Wall Street comprising the investing institutions such as mutual funds, pension funds and insurance firms that tends to buy large portions of securities for money management purposes (Investopedia, 2007).

FIX
Financial Information eXchange

FIX Protocol
The **Financial Information eXchange (FIX) Protocol** is a messaging standard developed specifically for the real-time electronic exchange of securities transactions. FIX is a public-domain specification owned and maintained by FIX Protocol, Ltd (FIX Protocol Limited, 2007).

FPL
FIX Protocol, Ltd. FPL is the governing body of the FIX Protocol.

Instrument
An instrument in this document refers to a financial instrument that would be traded between 2 parties, in the case of FIXimulator, they are equity instruments.

QuickFIX
QuickFIX is a full-featured open source FIX engine, currently compatible with the FIX 4.0-4.4 spec. It runs on Windows, Linux, Solaris, FreeBSD and Mac OS X. API's are available for C++, Java, .NET, Python and Ruby (QuickFIX, 2007).

QuickFIX/J
QuickFIX/J is a full featured messaging engine for the FIX protocol. It is a 100% Java open source implementation of the popular C++ QuickFIX engine (QuickFIX/J, 2007).

Sell side
Sell side entities, which provide recommendations for upgrades, downgrades, target prices and opinions to the public market. Together, the buy side and sell side make up both sides of Wall Street (Investopedia, 2007).

References

FIX Protocol Limited (2007). Implementation Guide. http://fixprotocol.org/implementation-guide/introduction.shtml, retrieved April 2007.

FIX Protocol Limited (2007). Specification Document with Errata as of 20010501 and Itemized Errata. http://fixprotocol.org/specifications/fix4.2spec, retrieved April 2007.

FIX Protocol Limited (2007). What is FIX? http://fixprotocol.org/what-is-fix.shtml, retrieved April 2007.

FIX Protocol Limited (2008). FIX-related products and services. http://fixprotocol.org/products/, retrieved August 2008.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley.

Investopedia Inc. (2007). Buy Side. http://www.investopedia.com/terms/b/buyside.asp, retrieved April 2007.

Investopedia Inc. (2007). Sell Side. http://www.investopedia.com/terms/s/sellside.asp, retrieved April 2007.

NetBeans (n. d.). Swing GUI Builder (formerly Project Matisse) http://www.netbeans.org/features/java/swing.html, retrieved August 2008.

QuickFIX (2001 – 2006). QuickFIX. http://www.quickfixengine.org/, retrieved April 2007.

QuickFIX/J (2007). 100% Java Open Source FIX Engine. http://www.quickfixj.org/, retrieved April 2007.

QuickFIX/J (2007). Example Applications, Banzai. http://www.quickfixj.org/quickfixj/usermanual/usage/examples.html, retrieved April 2007.

Sun Microsystems, Inc. (2006). Class GroupLayout. http://java.sun.com/javase/6/docs/api/javax/swing/GroupLayout.html, retrieved August 2008.

Sun Microsystems, Inc. (2002). Model – View – Controller. http://java.sun.com/blueprints/patterns/MVC.html, retrieved August 9, 2008.

Appendix 1 Use Cases

The use cases that follow are all based on the UML diagram in Figure 20.

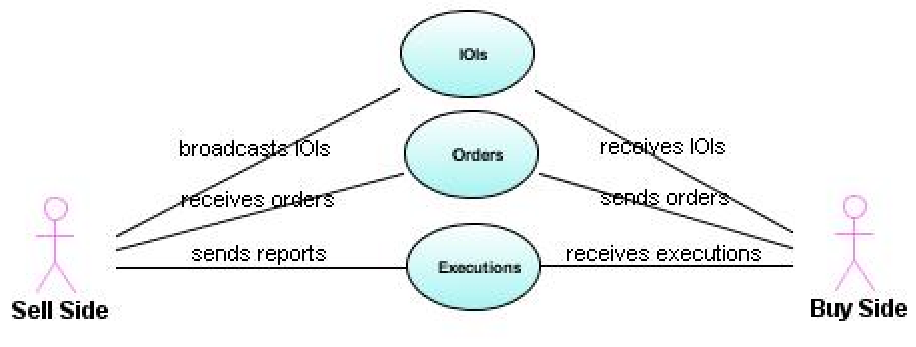


Figure 20 UML diagram of use cases

The first 2 use cases deal with the load testing of applications. Use case #1 is an example of a sell side broadcasting IOIs to the buy side as shown above. Receiving buy-side applications are often tasked with receiving large number of these messages. This case is an extension of the interaction that evaluates receiving system capacities. Use case #2 is another load test scenario where a buy side is tasked with receiving a large number of executions. Often the number of these reports present challenges to buy-side systems. Test case #3 is an adaptation of the state change matrices specified in Appendix D of the FIX 4.2 specification. This serves as a functional use case to verify that a buy-side system is able to correctly accommodate the statuses being reported by the sell side. The last use case addresses the various ways in which communication can be routed between the parties.

Use Case #1

Name: Load testing a buy-side OMS with IOIs

Summary: Use FIXimulator to load a receiving application with IOIs. This can be used to see how systems react to large volumes of IOIs received by their systems.

Preconditions: FIXimulator needs to be connected to the receiving system.

Triggers: The automated IOI sender needs to be started in FIXimulator.

Basic course of events:

1. Connect receiving system to FIXimulator
2. Start the automated IOI sender
3. Adjust rate as needed
4. Stop automated IOI sender when completed

Alternative paths:

1. Connect multiple instances of FIXimulator to receiving system for greater load
2. Start each automated IOI sender
3. Adjust rates as needed
4. Stop automated IOI senders when completed

Postconditions:

The receiving system was delivered a large number of IOIs as required by the IOI load test plan.

Author: Zoltan Feledy, August 2008

Use Case #2

Name: Load testing a buy-side OMS with executions.

Summary: Use FIXimulator to load a receiving application by automatically filling orders. This can be used to see how systems react to large volumes of fills received by their systems.

Preconditions: FIXimulator needs to be connected to the receiving system.

Triggers: The automated Executor needs to be started in FIXimulator.

Basic course of events:

1. Connect receiving system to FIXimulator
2. Start the automated Executor
3. Adjust number of fills per order as needed
4. Stop automated Executor when completed

Alternative paths:

1. Connect multiple instances of FIXimulator to receiving system for greater load
2. Start each automated Executor
3. Adjust number of fills per order as needed
4. Stop automated Executor when completed

Postconditions:

The receiving system was delivered a large number of fills as required by the load test plan.

Author: Zoltan Feledy, August 2008

Use Case #3

Name: Functional testing

Summary: Follow the 37 functional test cases listed in Appendix D of the FIX 4.2 specification. The below steps will describe what is required to carry out test case D1 in the specification.

Preconditions: FIXimulator needs to be connected to the receiving system.

Triggers: None.

Basic course of events:

1. Order is received by FIXimulator
2. Order is rejected by FIXimulator

Alternative paths:

1. Order is received by FIXimulator
2. Order is acknowledged by FIXimulator
3. Order is rejected by FIXimulator

1. Order for 10000 shares is received by FIXimulator
2. Order is acknowledged by FIXimulator
3. 2000 shares are filled by FIXimulator (partial fill)
4. 1000 shares are filled by FIXimulator (partial fill)
5. 7000 shares are filled by FIXimulator to completely fill the original order

Postconditions: Order appears in the receiving system as filled.

Author: Adapted from the FIX 4.2 Specification by Zoltan Feledy, August 2008

Use Case #4

Name: Routing configurations

Summary: FIXimulator can be configured for all available levels of routing. Configure each routing scheme to ensure that all are processed as needed by the receiving system.

Preconditions: FIXimulator is configured for each routing scheme.

Triggers: FIXimulator is connected with the appropriate routing scheme.

Basic course of events:

1. Configure FIXimulator as a direct connection (default)
2. Carry out the above functional test to ensure that the receiving system processes the messages as needed.

Alternative paths:

1. Configure FIXimulator to act as a third party connection by enabling the "Send OnBehalfOfCompID (115)" option in the settings and ensure that the appropriate value is configured in the settings.
2. Carry out the above functional test to ensure that the receiving system processes the messages as needed.

1. Configure FIXimulator to act as a third party connection by enabling the "Send OnBehalfOfSubID (116)" option in the settings and ensure that the appropriate value is configured in the settings.
2. Carry out the above functional test to ensure that the receiving system processes the messages as needed.

Postconditions: The receiving system has received messages from the appropriate end destinations.

Author: Zoltan Feledy, August 2008

Appendix 2 Application Code

Java Code

Execution.java

```
/*
 * File     : Execution.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is a basic Execution object that is used to
 *            create and store execution details.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.core;

public class Execution implements Cloneable {
    private static int nextID = 1;
    private Order order;
    private boolean DKd = false;
    private String ID = null;
    private String refID = null;
    private char execType;
    private char execTranType;
    private double lastShares = 0.0;
    private double lastPx = 0.0;
    private double leavesQty = 0.0;
    private double cumQty = 0.0;
    private double avgPx = 0.0;

    @Override
    public Execution clone() {
        try {
            Execution execution = (Execution)super.clone();
            execution.setRefID(getID());
            execution.setID(generateID());
            execution.setDKd(false);
            return execution;
        } catch(CloneNotSupportedException e) {}
        return null;
    }

    public Execution( Order order ) {
        ID = generateID();
        this.order = order;
    }

    public String generateID() {
        return "E" + Long.valueOf(
                System.currentTimeMillis()+(nextID++)).toString();
    }

    public String getID() {
        return ID;
    }

    public void setID(String ID) {
        this.ID = ID;
    }

    public boolean isDKd() {
        return DKd;
    }

    public void setDKd(boolean DKd) {
        this.DKd = DKd;
    }

    public double getAvgPx() {
        return avgPx;
    }

    public void setAvgPx(double avgPx) {
        this.avgPx = avgPx;
    }

    public double getCumQty() {
        return cumQty;
    }

    public void setCumQty(double cumQty) {
        this.cumQty = cumQty;
    }

    public String getExecTranType() {
        if (execTranType == '0') return "New";
        if (execTranType == '1') return "Cancel";
        if (execTranType == '2') return "Correct";
        if (execTranType == '3') return "Status";
        return "<UNKOWN>";
    }

    public char getFIXExecTranType() {
        return execTranType;
    }

    public void setExecTranType(char execTranType) {
        this.execTranType = execTranType;
    }
```

```
public String getExecType() {
    if (execType == '0') return "New";
    if (execType == '1') return "Partial fill";
    if (execType == '2') return "Fill";
    if (execType == '3') return "Done for day";
    if (execType == '4') return "Canceled";
    if (execType == '5') return "Replace";
    if (execType == '6') return "Pending Cancel";
    if (execType == '7') return "Stopped";
    if (execType == '8') return "Rejected";
    if (execType == '9') return "Suspended";
    if (execType == 'A') return "Pending New";
    if (execType == 'B') return "Calculated";
    if (execType == 'C') return "Expired";
    if (execType == 'D') return "Restated";
    if (execType == 'E') return "Pending Replace";
    return "<UNKNOWN>";
}

public char getFIXExecType() {
    return execType;
}

public void setExecType(char execType) {
    this.execType = execType;
}

public double getLastPx() {
    return lastPx;
}

public void setLastPx(double lastPx) {
    this.lastPx = lastPx;
}

public double getLastShares() {
    return lastShares;
}

public void setLastShares(double lastShares) {
    this.lastShares = lastShares;
}

public double getLeavesQty() {
    return leavesQty;
}

public void setLeavesQty(double leavesQty) {
    this.leavesQty = leavesQty;
}

public Order getOrder() {
    return order;
}

public void setOrder(Order order) {
    this.order = order;
    }

    public String getRefID() {
        return refID;
    }

    public void setRefID(String refID) {
        this.refID = refID;
    }
}
```

ExecutionSet.java

```
/*
 * File     : ExecutionSet.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is a Set of Execution objects with a utility
 *            methods to access the individual executions.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.core;

import java.util.ArrayList;
import java.util.Iterator;
import edu.harvard.fas.zfeledy.fiximulator.ui.ExecutionTableModel;

public class ExecutionSet {
    private ArrayList<Execution> executions = new
ArrayList<Execution>();
    private ExecutionTableModel executionTableModel = null;

    public ExecutionSet() {}

    public void add ( Execution execution ) {
        executions.add( execution );
        int limit = 50;
        try {
            limit = (int)FIXimulator.getApplication().getSettings()
                    .getLong("FIXimulatorCachedObjects");
        } catch ( Exception e ) {}
        while ( executions.size() > limit ) {
            executions.remove(0);
        }
        executionTableModel.update();
    }

    public void update () {
        executionTableModel.update();
    }

    public void addCallback(ExecutionTableModel executionTableModel){
        this.executionTableModel = executionTableModel;
    }
    public int getCount() {
        return executions.size();
    }

    public Execution getExecution( int i ) {
        return executions.get( i );
    }

    public Execution getExecution( String id ) {
        Iterator<Execution> iterator = executions.iterator();
        while ( iterator.hasNext() ){
            Execution execution = iterator.next();
            if ( execution.getID().equals(id) )
                return execution;
        }
        return null;
    }
}
```

FIXimulator.java

```
/*
 * File     : FIXimulator.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This is the class that initializes the application.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.core;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;

import quickfix.Acceptor;
import quickfix.CompositeLogFactory;
import quickfix.ConfigError;
import quickfix.DefaultMessageFactory;
import quickfix.FieldConvertError;
import quickfix.FileLogFactory;
import quickfix.FileStoreFactory;
import quickfix.JdbcLogFactory;
import quickfix.LogFactory;
import quickfix.MessageFactory;
import quickfix.MessageStoreFactory;
import quickfix.ScreenLogFactory;
import quickfix.SessionSettings;
import quickfix.SocketAcceptor;

public class FIXimulator {
    private static final long serialVersionUID = 1L;
    private Acceptor acceptor = null;
    private static FIXimulatorApplication application = null;
    private static InstrumentSet instruments = null;
    private static LogMessageSet messages = null;

    public FIXimulator() {
        InputStream inputStream = null;
        try {
            inputStream = new BufferedInputStream(
                                new FileInputStream(
                                new File( "config/FIXimulator.cfg" )));
        } catch (FileNotFoundException exception) {
            exception.printStackTrace();
            System.exit(0);
        }
        instruments = new InstrumentSet(new
File("config/instruments.xml"));
        messages = new LogMessageSet();
        try {
            SessionSettings settings = new SessionSettings( inputStream
);
            application = new FIXimulatorApplication( settings,
messages );
            MessageStoreFactory messageStoreFactory =
                    new FileStoreFactory( settings );
            boolean logToFile = false;
            boolean logToDB = false;
            LogFactory logFactory;
            try {
                logToFile = settings.getBool("FIXimulatorLogToFile");
                logToDB = settings.getBool("FIXimulatorLogToDB");
            } catch (FieldConvertError ex) {}
            if ( logToFile && logToDB ) {
                logFactory = new CompositeLogFactory(
                    new LogFactory[] { new ScreenLogFactory(settings),
                                       new FileLogFactory(settings),
                                       new JdbcLogFactory(settings)});
            } else if ( logToFile ) {
                logFactory = new CompositeLogFactory(
                    new LogFactory[] { new ScreenLogFactory(settings),
                                       new FileLogFactory(settings)});
            } else if ( logToDB ) {
                logFactory = new CompositeLogFactory(
                    new LogFactory[] { new ScreenLogFactory(settings),
                                       new JdbcLogFactory(settings)});
            } else {
                logFactory = new ScreenLogFactory(settings);
            }
            MessageFactory messageFactory = new
DefaultMessageFactory();
            acceptor = new SocketAcceptor
                    ( application, messageStoreFactory,
                                settings, logFactory, messageFactory );
        } catch (ConfigError e) {
            e.printStackTrace();
        }
    }
    public static InstrumentSet getInstruments() {
        return instruments;
    }

    public static FIXimulatorApplication getApplication() {
        return application;
    }

    public static LogMessageSet getMessageSet() {
        return messages;
    }

    public void start() {
        try {
            acceptor.start();
        } catch ( Exception e ) {
            System.out.println( e );
        }
    }
}
```

FIXimulatorApplication.java

```
/*
 * File     : FIXimulatorApplication.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This is the application class that contains all the
 *            logic for message handling.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.core;

import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.util.Date;
import java.util.Random;

import javax.swing.JLabel;
import quickfix.Application;
import quickfix.DataDictionary;
import quickfix.DoNotSend;
import quickfix.FieldNotFound;
import quickfix.IncorrectDataFormat;
import quickfix.IncorrectTagValue;
import quickfix.Message;
import quickfix.MessageCracker;
import quickfix.RejectLogon;
import quickfix.Session;
import quickfix.SessionID;
import quickfix.SessionNotFound;
import quickfix.SessionSettings;
import quickfix.UnsupportedMessageType;
import quickfix.field.AvgPx;
import quickfix.field.ClOrdID;
import quickfix.field.CumQty;
import quickfix.field.Currency;
import quickfix.field.CxlRejResponseTo;
import quickfix.field.ExecID;
import quickfix.field.ExecRefID;
import quickfix.field.ExecTransType;
import quickfix.field.ExecType;
import quickfix.field.IDSource;
import quickfix.field.IOINaturalFlag;
import quickfix.field.IOIRefID;
import quickfix.field.IOIShares;
import quickfix.field.IOITransType;
import quickfix.field.IOIid;
import quickfix.field.LastPx;
import quickfix.field.LastShares;
import quickfix.field.LeavesQty;
import quickfix.field.OnBehalfOfCompID;
import quickfix.field.OnBehalfOfSubID;
import quickfix.field.OrdStatus;
import quickfix.field.OrderID;
import quickfix.field.OrderQty;
import quickfix.field.OrigClOrdID;
import quickfix.field.Price;
import quickfix.field.SecurityDesc;
import quickfix.field.SecurityID;
import quickfix.field.Side;
import quickfix.field.Symbol;
import quickfix.field.ValidUntilTime;
import quickfix.fix42.Message.Header;

public class FIXimulatorApplication extends MessageCracker
                                    implements Application {
    private boolean connected;
    private JLabel connectedStatus;
    private JLabel ioiSenderStatus;
    private JLabel executorStatus;
    private boolean ioiSenderStarted;
    private boolean executorStarted;
    private IOIsender ioiSender;
    private Thread ioiSenderThread;
    private Executor executor;
    private Thread executorThread;
    private LogMessageSet messages;
    private SessionSettings settings;
    private SessionID currentSession;
    private DataDictionary dictionary;
    private Random random = new Random();
    private IOIset iois = null;
    private OrderSet orders = null;
    private ExecutionSet executions = null;

    public FIXimulatorApplication( SessionSettings settings,
            LogMessageSet messages ){
            this.settings = settings;
            this.messages = messages;
            iois = new IOIset();
            orders = new OrderSet();
            executions = new ExecutionSet();
    }

    public void onCreate( SessionID sessionID ) {}

    public void onLogon( SessionID sessionID ) {
        connected = true;
        currentSession = sessionID;
        dictionary =
Session.lookupSession(currentSession).getDataDictionary();
        if (connectedStatus != null)
            connectedStatus.setIcon(new
javax.swing.ImageIcon(getClass()

.getResource("/edu/harvard/fas/zfeledy/fiximulator/ui/green.gif")));
    }

    public void onLogout( SessionID sessionID ) {
      connected = false;
      currentSession = null;
        connectedStatus.setIcon(new javax.swing.ImageIcon(getClass()

.getResource("/edu/harvard/fas/zfeledy/fiximulator/ui/red.gif")));
    }

    // IndicationofInterest handling
    @Override
    public void onMessage( quickfix.fix42.IndicationofInterest message,
            SessionID sessionID )
      throws FieldNotFound, UnsupportedMessageType, IncorrectTagValue
{}

    // NewOrderSingle handling
    @Override
    public void onMessage( quickfix.fix42.NewOrderSingle message,
            SessionID sessionID )
      throws FieldNotFound, UnsupportedMessageType, IncorrectTagValue {
        Order order = new Order( message );
        order.setReceivedOrder( true );
        if ( executorStarted ) {
            orders.add( order, true );
            executorThread.interrupt();
        } else {
            orders.add( order, false );
            boolean autoAck = false;
            try {
                autoAck =
settings.getBool("FIXimulatorAutoAcknowledge");
            } catch ( Exception e ) {}
            if ( autoAck ) {
                acknowledge( order );
            }
        }
    }

    // OrderCancelRequest handling
    @Override
    public void onMessage( quickfix.fix42.OrderCancelRequest message,
            SessionID sessionID )
      throws FieldNotFound, UnsupportedMessageType, IncorrectTagValue {
        Order order = new Order( message );
        order.setReceivedCancel( true );
        orders.add( order, false );
        boolean autoPending = false;
        boolean autoCancel = false;
        try {
            autoPending =
settings.getBool("FIXimulatorAutoPendingCancel");
            autoCancel = settings.getBool("FIXimulatorAutoCancel");
        } catch ( Exception e ) {}
        if ( autoPending ) {
            pendingCancel(order);
        }
        if ( autoCancel ) {
            cancel(order);
        }
    }

    // OrderReplaceRequest handling
    @Override
    public void onMessage( quickfix.fix42.OrderCancelReplaceRequest
message,
            SessionID sessionID )
      throws FieldNotFound, UnsupportedMessageType, IncorrectTagValue {
        Order order = new Order( message );
        order.setReceivedReplace( true );
        orders.add( order, false );
        boolean autoPending = false;
        boolean autoCancel = false;
        try {
            autoPending =
settings.getBool("FIXimulatorAutoPendingReplace");
            autoCancel = settings.getBool("FIXimulatorAutoReplace");
        } catch ( Exception e ) {}
        if ( autoPending ) {
            pendingReplace(order);
        }
        if ( autoCancel ) {
            replace(order);
        }
    }

    // OrderCancelReject handling
    @Override
    public void onMessage( quickfix.fix42.OrderCancelReject message,
            SessionID sessionID )
      throws FieldNotFound, UnsupportedMessageType, IncorrectTagValue
{}
```

```
    // ExecutionReport handling
    @Override
    public void onMessage( quickfix.fix42.ExecutionReport message,
            SessionID sessionID )
      throws FieldNotFound, UnsupportedMessageType, IncorrectTagValue
{}

    @Override
    public void onMessage( quickfix.fix42.DontKnowTrade message,
            SessionID sessionID )
      throws FieldNotFound, UnsupportedMessageType, IncorrectTagValue {

        try {
            ExecID execID = new ExecID();
            message.get(execID);
            Execution execution =

executions.getExecution(execID.getValue().toString());
            execution.setDKd(true);
            executions.update();
        } catch (FieldNotFound ex) {}
    }

    public void fromApp( Message message, SessionID sessionID )
        throws FieldNotFound, IncorrectDataFormat,
            IncorrectTagValue, UnsupportedMessageType {
      messages.add(message, true, dictionary, sessionID);
        crack(message, sessionID);
    }

    public void toApp( Message message, SessionID sessionID ) throws
DoNotSend {
      try {
            messages.add(message, false, dictionary, sessionID);
            crack(message, sessionID);
        } catch (Exception e) {     e.printStackTrace(); }
    }

    public void fromAdmin( Message message, SessionID sessionID )
      throws
        FieldNotFound, IncorrectDataFormat, IncorrectTagValue,
RejectLogon {}

    public void toAdmin( Message message, SessionID sessionID ) {}

    public void addStatusCallbacks ( JLabel connectedStatus,
            JLabel ioiSenderStatus, JLabel executorStatus ) {
        this.connectedStatus = connectedStatus;
        this.ioiSenderStatus = ioiSenderStatus;
        this.executorStatus = executorStatus;
    }

    public boolean getConnectionStatus(){
        return connected;
    }

    public IOIset getIOIs() {
```

```
    return iois;
}

public OrderSet getOrders() {
    return orders;
}

public ExecutionSet getExecutions() {
    return executions;
}

public SessionSettings getSettings() {
    return settings;
}

public void saveSettings() {
    try {
        OutputStream outputStream =
                new BufferedOutputStream(
                new FileOutputStream(
                new File("config/FIXimulator.cfg")));
        settings.toStream(outputStream);
    } catch (FileNotFoundException ex) {
        ex.printStackTrace();
    }
}

// Message handling methods
public void acknowledge( Order order ) {
    Execution acknowledgement = new Execution(order);
    order.setStatus(OrdStatus.NEW);
    acknowledgement.setExecType(ExecType.NEW);
    acknowledgement.setExecTranType(ExecTransType.NEW);
    acknowledgement.setLeavesQty(order.getOpen());
    sendExecution(acknowledgement);
    order.setReceivedOrder(false);
    orders.update();
}

public void reject( Order order ) {
    Execution reject = new Execution(order);
    order.setStatus(OrdStatus.REJECTED);
    reject.setExecType(ExecType.REJECTED);
    reject.setExecTranType(ExecTransType.NEW);
    reject.setLeavesQty(order.getOpen());
    sendExecution(reject);
    order.setReceivedOrder(false);
    orders.update();
}

public void dfd( Order order ) {
    Execution dfd = new Execution(order);
    order.setStatus(OrdStatus.DONE_FOR_DAY);
    dfd.setExecType(ExecType.DONE_FOR_DAY);
    dfd.setExecTranType(ExecTransType.NEW);
    dfd.setLeavesQty(order.getOpen());
    dfd.setCumQty(order.getExecuted());
```

```
        dfd.setAvgPx(order.getAvgPx());
        sendExecution(dfd);
        orders.update();
    }

    public void pendingCancel( Order order ) {
        Execution pending = new Execution(order);
        order.setStatus(OrdStatus.PENDING_CANCEL);
        pending.setExecType(ExecType.PENDING_CANCEL);
        pending.setExecTranType(ExecTransType.NEW);
        pending.setLeavesQty(order.getOpen());
        pending.setCumQty(order.getExecuted());
        pending.setAvgPx(order.getAvgPx());
        sendExecution(pending);
        order.setReceivedCancel(false);
        orders.update();
    }

    public void cancel( Order order ) {
        Execution cancel = new Execution(order);
        order.setStatus(OrdStatus.CANCELED);
        cancel.setExecType(ExecType.CANCELED);
        cancel.setExecTranType(ExecTransType.NEW);
        cancel.setLeavesQty(order.getOpen());
        cancel.setCumQty(order.getExecuted());
        cancel.setAvgPx(order.getAvgPx());
        sendExecution(cancel);
        order.setReceivedCancel(false);
        orders.update();
    }

    public void rejectCancelReplace( Order order, boolean cancel ) {
        order.setReceivedCancel(false);
        order.setReceivedReplace(false);
        // *** Required fields ***
        // OrderID (37)
        OrderID orderID = new OrderID(order.getID());

        ClOrdID clientID = new ClOrdID(order.getClientID());

        OrigClOrdID origClientID = new
OrigClOrdID(order.getOrigClientID());

        // OrdStatus (39) Status as a result of this report
        if ( order.getStatus().equals("<UNKNOWN>") )
            order.setStatus(OrdStatus.NEW);
        OrdStatus ordStatus = new OrdStatus(order.getFIXStatus());

        CxlRejResponseTo responseTo = new CxlRejResponseTo();
        if ( cancel ) {
            responseTo.setValue(CxlRejResponseTo.ORDER_CANCEL_REQUEST);
        } else {

responseTo.setValue(CxlRejResponseTo.ORDER_CANCEL_REPLACE_REQUEST);
        }

        // Construct OrderCancelReject message from required fields
        quickfix.fix42.OrderCancelReject rejectMessage =
                 new quickfix.fix42.OrderCancelReject(
                    orderID,
                    clientID,
                    origClientID,
                    ordStatus,
                    responseTo);

        // *** Send message ***
        sendMessage(rejectMessage);
        orders.update();
    }

    public void pendingReplace( Order order ) {
        Execution pending = new Execution(order);
        order.setStatus(OrdStatus.PENDING_REPLACE);
        pending.setExecType(ExecType.PENDING_REPLACE);
        pending.setExecTranType(ExecTransType.NEW);
        pending.setLeavesQty(order.getOpen());
        pending.setCumQty(order.getExecuted());
        pending.setAvgPx(order.getAvgPx());
        order.setReceivedReplace(false);
        sendExecution(pending);
        orders.update();
    }

    public void replace( Order order ) {
        Execution replace = new Execution(order);
        order.setStatus(OrdStatus.REPLACED);
        replace.setExecType(ExecType.REPLACE);
        replace.setExecTranType(ExecTransType.NEW);
        replace.setLeavesQty(order.getOpen());
        replace.setCumQty(order.getExecuted());
        replace.setAvgPx(order.getAvgPx());
        order.setReceivedReplace(false);
        sendExecution(replace);
        orders.update();
    }

    public void execute( Execution execution ){
        Order order = execution.getOrder();
        double fillQty = execution.getLastShares();
        double fillPrice = execution.getLastPx();
        double open = order.getOpen();
        // partial fill
        if ( fillQty < open ) {
            order.setOpen( open - fillQty );
            order.setStatus(OrdStatus.PARTIALLY_FILLED);
            execution.setExecType(ExecType.PARTIAL_FILL);
            // full or over execution
        } else {
            order.setOpen(0);
            order.setStatus(OrdStatus.FILLED);
            execution.setExecType(ExecType.FILL);
        }
        double avgPx =(order.getAvgPx() * order.getExecuted()
                     + fillPrice * fillQty)
                 /(order.getExecuted() + fillQty );
        order.setAvgPx(avgPx);
        order.setExecuted(order.getExecuted() + fillQty);
        orders.update();
        // update execution
        execution.setExecTranType(ExecTransType.NEW);
        execution.setLeavesQty(order.getOpen());
        execution.setCumQty(order.getExecuted());
        execution.setAvgPx(avgPx);
        sendExecution(execution);
    }

    public void bust( Execution execution ){
        Execution bust = execution.clone();
        Order order = execution.getOrder();
        double fillQty = execution.getLastShares();
        double fillPrice = execution.getLastPx();
        double executed = order.getExecuted();
        // partial fill
        if ( fillQty < executed ) {
            order.setOpen( executed - fillQty );
            order.setStatus(OrdStatus.PARTIALLY_FILLED);
            double avgPx =(order.getAvgPx() * executed
                        - fillPrice * fillQty)
                        /(order.getExecuted() - fillQty );
            order.setAvgPx(avgPx);
            order.setExecuted(order.getExecuted() - fillQty);
            // full or over execution
        } else {
            order.setOpen(order.getQuantity());
            order.setStatus(OrdStatus.NEW);
            order.setAvgPx(0);
            order.setExecuted(0);
        }
        orders.update();
        // update execution
        bust.setExecTranType(ExecTransType.CANCEL);
        bust.setLeavesQty(order.getOpen());
        bust.setCumQty(order.getExecuted());
        bust.setAvgPx(order.getAvgPx());
        sendExecution(bust);
    }

    public void correct( Execution correction ){
        Order order = correction.getOrder();
        Execution original =
executions.getExecution(correction.getRefID());

        double fillQty = correction.getLastShares();
        double oldQty = original.getLastShares();

        double fillPrice = correction.getLastPx();
        double oldPrice = original.getLastPx();

        double executed = order.getExecuted();
        double ordered = order.getQuantity();
        double newCumQty = executed - oldQty + fillQty;
        double avgPx =(order.getAvgPx() * executed
                     - oldPrice * oldQty
                     + fillPrice * fillQty)
                     / newCumQty;

        // partial fill
        if ( newCumQty < ordered ) {
            order.setOpen( ordered - newCumQty );
            order.setStatus(OrdStatus.PARTIALLY_FILLED);
        // full or over execution
        } else {
            order.setOpen(0);
            order.setStatus(OrdStatus.FILLED);
        }

        order.setAvgPx(avgPx);
        order.setExecuted(newCumQty);
        orders.update();

        // update execution
        correction.setExecTranType(ExecTransType.CORRECT);
        correction.setLeavesQty(order.getOpen());
        correction.setCumQty(order.getExecuted());
        correction.setAvgPx(order.getAvgPx());
        sendExecution(correction);
    }

    // Message sending methods
    public void sendMessage( Message message ) {
        String oboCompID = "<UNKNOWN>";
        String oboSubID = "<UNKNOWN>";
        boolean sendoboCompID = false;
        boolean sendoboSubID = false;

        try {
            oboCompID = settings.getString(currentSession,
"OnBehalfOfCompID");
            oboSubID = settings.getString(currentSession,
"OnBehalfOfSubID");
            sendoboCompID =
settings.getBool("FIXimulatorSendOnBehalfOfCompID");
            sendoboSubID =
settings.getBool("FIXimulatorSendOnBehalfOfSubID");
        } catch ( Exception e ) {}

        // Add OnBehalfOfCompID
        if ( sendoboCompID && !oboCompID.equals("") ) {
            OnBehalfOfCompID onBehalfOfCompID = new
OnBehalfOfCompID(oboCompID);
            Header header = (Header) message.getHeader();
            header.set( onBehalfOfCompID );
        }

        // Add OnBehalfOfSubID
        if ( sendoboSubID && !oboSubID.equals("") ) {
            OnBehalfOfSubID onBehalfOfSubID = new
OnBehalfOfSubID(oboSubID);
            Header header = (Header) message.getHeader();
            header.set( onBehalfOfSubID );
        }

        // Send actual message
        try {
            Session.sendToTarget( message, currentSession );
        } catch ( SessionNotFound e ) { e.printStackTrace(); }
    }

    public void sendIOI( IOI ioi ) {
        // *** Required fields ***
        // IOIid
        IOIid ioiID = new IOIid( ioi.getID() );

        // IOITransType
        IOITransType ioiType = null;
        if ( ioi.getType().equals("NEW") )
            ioiType = new IOITransType( IOITransType.NEW );
        if ( ioi.getType().equals("CANCEL") )
            ioiType = new IOITransType( IOITransType.CANCEL );
        if ( ioi.getType().equals("REPLACE") )
            ioiType = new IOITransType( IOITransType.REPLACE );

        // Side
        Side side = null;
        if ( ioi.getSide().equals("BUY") ) side = new Side( Side.BUY );
        if ( ioi.getSide().equals("SELL") ) side = new Side( Side.SELL
);
        if ( ioi.getSide().equals("UNDISCLOSED") )
            side = new Side( Side.UNDISCLOSED );

        // IOIShares
        IOIShares shares = new IOIShares( ioi.getQuantity().toString()
);

        // Symbol
        Symbol symbol = new Symbol( ioi.getSymbol() );

        // Construct IOI from required fields
        quickfix.fix42.IndicationofInterest fixIOI =
            new quickfix.fix42.IndicationofInterest(
            ioiID, ioiType, symbol, side, shares);

        // *** Conditionally required fields ***
        // IOIRefID
        IOIRefID ioiRefID = null;
        if ( ioi.getType().equals("CANCEL") ||
ioi.getType().equals("REPLACE")){
            ioiRefID = new IOIRefID( ioi.getRefID() );
            fixIOI.set(ioiRefID);
        }

        // *** Optional fields ***
        // SecurityID
        SecurityID securityID = new SecurityID( ioi.getSecurityID() );
        fixIOI.set( securityID );

        // IDSource
        IDSource idSource = null;
        if (ioi.getIDSource().equals("TICKER"))
            idSource = new IDSource( IDSource.EXCHANGE_SYMBOL );
        if (ioi.getIDSource().equals("RIC"))
            idSource = new IDSource( IDSource.RIC_CODE );
        if (ioi.getIDSource().equals("SEDOL"))
            idSource = new IDSource( IDSource.SEDOL );
        if (ioi.getIDSource().equals("CUSIP"))
            idSource = new IDSource( IDSource.CUSIP );
        if (ioi.getIDSource().equals("UNKOWN"))
            idSource = new IDSource( "100" );
        fixIOI.set( idSource );

        // Price
        Price price = new Price( ioi.getPrice() );
        fixIOI.set( price );

        // IOINaturalFlag
        IOINaturalFlag ioiNaturalFlag = new IOINaturalFlag();
        if ( ioi.getNatural().equals("YES") )
            ioiNaturalFlag.setValue( true );
        if ( ioi.getNatural().equals("NO") )
            ioiNaturalFlag.setValue( false );
        fixIOI.set( ioiNaturalFlag );

        // SecurityDesc
        Instrument instrument =

FIXimulator.getInstruments().getInstrument(ioi.getSymbol());
        String name = "Unknown security";
        if ( instrument != null ) name = instrument.getName();
        SecurityDesc desc = new SecurityDesc( name );
        fixIOI.set( desc );

        // ValidUntilTime
        int minutes = 30;
        long expiry = new Date().getTime() + 1000 * 60 * minutes;
        Date validUntil = new Date( expiry );
        ValidUntilTime validTime = new ValidUntilTime( validUntil );
        fixIOI.set( validTime );

        //Currency
        Currency currency = new Currency( "USD" );
        fixIOI.set( currency );

        // *** Send message ***
        sendMessage(fixIOI);
        iois.add(ioi);
    }

    public void sendExecution( Execution execution ) {
        Order order = execution.getOrder();
        // *** Required fields ***
        // OrderID (37)
        OrderID orderID = new OrderID(order.getID());

        // ExecID (17)
        ExecID execID = new ExecID(execution.getID());

        // ExecTransType (20)
        ExecTransType execTransType =
                new ExecTransType(execution.getFIXExecTranType());

        // ExecType (150) Status of this report
        ExecType execType = new ExecType(execution.getFIXExecType());

        // OrdStatus (39) Status as a result of this report
        OrdStatus ordStatus =
                new OrdStatus(execution.getOrder().getFIXStatus());

        // Symbol (55)
        Symbol symbol = new Symbol(execution.getOrder().getSymbol());

        //  Side (54)
        Side side = new Side(execution.getOrder().getFIXSide());

        // LeavesQty ()
        LeavesQty leavesQty = new LeavesQty(execution.getLeavesQty());

        // CumQty ()
        CumQty cumQty = new CumQty(execution.getCumQty());

        // AvgPx ()
        AvgPx avgPx = new AvgPx(execution.getAvgPx());

        // Construct Execution Report from required fields
        quickfix.fix42.ExecutionReport executionReport =
                new quickfix.fix42.ExecutionReport(
                    orderID,
                    execID,
                    execTransType,
                    execType,
                    ordStatus,
                    symbol,
                    side,
                    leavesQty,
                    cumQty,
                    avgPx);

        // *** Conditional fields ***
        if(execution.getRefID() != null) {
            executionReport.set(
                    new ExecRefID(execution.getRefID()));
        }

        // *** Optional fields ***
        executionReport.set(new
ClOrdID(execution.getOrder().getClientID()));
```

```
        executionReport.set(new
OrderQty(execution.getOrder().getQuantity()));
        executionReport.set(new LastShares(execution.getLastShares()));
        executionReport.set(new LastPx(execution.getLastPx()));
        if( order.getSecurityID() != null
            && order.getIdSource()!= null) {
            executionReport.set(new SecurityID(order.getSecurityID()));
            executionReport.set(new IDSource(order.getIdSource()));
        }

        // *** Send message ***
        sendMessage(executionReport);
        executions.add(execution);
    }

    // IOI Sender methods
    public void startIOIsender(Integer delay, String symbol, String
securityID){
        try {
            ioiSender =  new IOIsender(delay, symbol, securityID);
            ioiSenderThread = new Thread(ioiSender);
            ioiSenderThread.start();
        } catch (Exception e) {e.printStackTrace();}
        if (connected && ioiSenderStarted)
            ioiSenderStatus.setIcon(new
javax.swing.ImageIcon(getClass()

.getResource("/edu/harvard/fas/zfeledy/fiximulator/ui/green.gif")));
    }

    public void stopIOIsender(){
      ioiSender.stopIOISender();
        ioiSenderThread.interrupt();
      try {
            ioiSenderThread.join();
        } catch (InterruptedException e) {e.printStackTrace();}
        ioiSenderStatus.setIcon(new javax.swing.ImageIcon(getClass()

.getResource("/edu/harvard/fas/zfeledy/fiximulator/ui/red.gif")));
    }

    public void setNewDelay(Integer delay) {
        if (ioiSenderStarted) ioiSender.setDelay(delay);
    }

    public void setNewSymbol( String identifier ) {
        if (ioiSenderStarted) ioiSender.setSymbol(identifier);
    }

    public void setNewSecurityID( String identifier ) {
        if (ioiSenderStarted) ioiSender.setSecurityID(identifier);
    }

    public class IOIsender implements Runnable {
      InstrumentSet instruments;
      private Integer delay;
      private String symbolValue = "";
      private String securityIDvalue = "";

      public IOIsender ( Integer delay, String symbol, String
securityID ) {
            instruments = FIXimulator.getInstruments();
            ioiSenderStarted = true;
            this.delay = delay;
            symbolValue = symbol;
            securityIDvalue = securityID;
      }

      public void stopIOISender() {
            ioiSenderStarted = false;
        }

      public void setDelay(Integer delay){
            this.delay = delay;
      }

      public void setSymbol( String identifier ) {
            symbolValue = identifier;
      }

      public void setSecurityID ( String identifier ) {
            securityIDvalue = identifier;
      }

        public void run() {
            while ( connected && ioiSenderStarted ) {
                sendRandomIOI();
                try {
                    Thread.sleep( delay.longValue() );
                } catch ( InterruptedException e ) {}
            }
            ioiSenderStatus.setIcon(new
javax.swing.ImageIcon(getClass()

.getResource("/edu/harvard/fas/zfeledy/fiximulator/ui/red.gif")));
        }

      public void sendRandomIOI() {
            Instrument instrument = instruments.randomInstrument();
            IOI ioi = new IOI();
            ioi.setType("NEW");

            // Side
            ioi.setSide("BUY");
            if ( random.nextBoolean() ) ioi.setSide("SELL");

            // IOIShares
            Integer quantity = new Integer( random.nextInt(1000) * 100
+ 100 );
            ioi.setQuantity( quantity );

            // Symbol
            String value = "";
            if (symbolValue.equals( "Ticker" ) ) value =
instrument.getTicker();
            if (symbolValue.equals( "RIC" ) ) value =
instrument.getRIC();
            if (symbolValue.equals( "Sedol" ) ) value =
instrument.getSedol();
            if (symbolValue.equals( "Cusip" ) ) value =
instrument.getCusip();
            if ( value.equals( "" ) ) value = "<MISSING>";
            ioi.setSymbol( value );
            Symbol symbol = new Symbol( ioi.getSymbol() );

            // *** Optional fields ***
            // SecurityID
            value = "";
            if (securityIDvalue.equals("Ticker"))
                value = instrument.getTicker();
            if (securityIDvalue.equals("RIC"))
                value = instrument.getRIC();
            if (securityIDvalue.equals("Sedol"))
                value = instrument.getSedol();
            if (securityIDvalue.equals("Cusip"))
                value = instrument.getCusip();
            if ( value.equals( "" ) )
                value = "<MISSING>";
            ioi.setSecurityID( value );

            // IDSource
            if ( securityIDvalue.equals("Ticker") )
ioi.setIDSource("TICKER");
            if ( securityIDvalue.equals("RIC") )
ioi.setIDSource("RIC");
            if ( securityIDvalue.equals("Sedol") )
ioi.setIDSource("SEDOL");
            if ( securityIDvalue.equals("Cusip") )
ioi.setIDSource("CUSIP");
            if ( ioi.getSecurityID().equals("<MISSING>") )
                ioi.setIDSource("UNKNOWN");

            // Price
            int pricePrecision = 4;
            try {
                pricePrecision =

(int)settings.getLong("FIXimulatorPricePrecision");
            } catch ( Exception e ) {}
            double factor = Math.pow( 10, pricePrecision );
            double price = Math.round(
                    random.nextDouble() * 100 * factor ) / factor;
            ioi.setPrice(price);

            // IOINaturalFlag
            ioi.setNatural("No");
            if ( random.nextBoolean() ) ioi.setNatural("Yes");

            sendIOI(ioi);
        }
```

```
    }

    // Executor methods
    public void startExecutor( Integer delay, Integer partials ) {
        try {
            executor =  new Executor( delay, partials );
            executorThread = new Thread(executor);
            executorThread.start();
        } catch (Exception e) {e.printStackTrace();}
        if (connected && executorStarted)
            executorStatus.setIcon(new javax.swing.ImageIcon(getClass()

.getResource("/edu/harvard/fas/zfeledy/fiximulator/ui/green.gif")));
    }

    public void stopExecutor(){
        executor.stopExecutor();
        executorThread.interrupt();
      try {
            executorThread.join();
        } catch (InterruptedException e) {e.printStackTrace();}
        executorStatus.setIcon(new javax.swing.ImageIcon(getClass()

.getResource("/edu/harvard/fas/zfeledy/fiximulator/ui/red.gif")));
    }

    public void setNewExecutorDelay( Integer delay ) {
        if (executorStarted) executor.setDelay(delay);
    }

    public void setNewExecutorPartials( Integer partials ) {
        if (executorStarted) executor.setPartials(partials);
    }

    public class Executor implements Runnable {
      InstrumentSet instruments;
      private Integer delay;
      private Integer partials;

        public Executor( Integer delay, Integer partials ) {
            instruments = FIXimulator.getInstruments();
            executorStarted = true;
            this.partials = partials;
            this.delay = delay;
        }

        public void run() {
            while ( connected && executorStarted ) {
                while ( orders.haveOrdersToFill() ) {
                    Order order = orders.getOrderToFill();
                    acknowledge(order);
                    fill(order);
                }
                // No orders to fill, check again in 5 seconds
                try {
                    Thread.sleep( 5000 );
                } catch ( InterruptedException e ) {}
            }
            executorStatus.setIcon(new javax.swing.ImageIcon(getClass()

.getResource("/edu/harvard/fas/zfeledy/fiximulator/ui/red.gif")));
        }

        public void stopExecutor(){
            executorStarted = false;
        }

        public void setDelay(Integer delay){
            this.delay = delay;
      }

        public void setPartials(Integer partials){
            this.partials = partials;
        }

        public void fill( Order order ) {
            double fillQty = Math.floor( order.getQuantity() / partials
);
            double fillPrice = 0.0;
            // try to look the price up from the instruments
            Instrument instrument =
                    instruments.getInstrument(order.getSymbol());
            if (instrument != null){
                fillPrice = Double.valueOf(instrument.getPrice());
            // use a random price
            } else {
                int pricePrecision = 4;
                try {
                    pricePrecision =

(int)settings.getLong("FIXimulatorPricePrecision");
                } catch ( Exception e ) {}
                double factor = Math.pow( 10, pricePrecision );
                fillPrice = Math.round(
                    random.nextDouble() * 100 * factor ) / factor;
            }

            if ( fillQty == 0 ) fillQty = 1;
            for (int i=0; i < partials; i++) {
                double open = order.getOpen();
                if ( open > 0 ) {
                    if ( fillQty < open && i != partials - 1) {
                        // send partial
                        // calculate fields
                        double priorQty = order.getExecuted();
                        double priorAvg = order.getAvgPx();
                        if (random.nextBoolean())
                            fillPrice += 0.01;
                        else
                            fillPrice -= 0.01;
                        double thisAvg = ((fillQty*fillPrice)
                                + (priorQty*priorAvg))
                                / (priorQty + fillQty);
                        int pricePrecision = 4;
                        try {
                            pricePrecision =

(int)settings.getLong("FIXimulatorPricePrecision");
                        } catch ( Exception e ) {}
                        double factor = Math.pow( 10, pricePrecision );
                        fillPrice = Math.round( fillPrice * factor ) /
factor;
                        thisAvg = Math.round( thisAvg * factor ) /
factor;
                       // update order
                        order.setOpen( open - fillQty );
                        order.setStatus(OrdStatus.PARTIALLY_FILLED);
                        order.setExecuted(order.getExecuted() +
fillQty);
                        order.setAvgPx(thisAvg);
                        orders.update();
                        // create execution
                        Execution partial = new Execution(order);
                        partial.setExecType(ExecType.PARTIAL_FILL);
                        partial.setExecTranType(ExecTransType.NEW);
                        partial.setLeavesQty(order.getOpen());
                        partial.setCumQty(order.getQuantity()-
order.getOpen());
                        partial.setAvgPx(thisAvg);
                        partial.setLastShares(fillQty);
                        partial.setLastPx(fillPrice);
                        sendExecution(partial);
                    } else {
                        // send full
                        fillQty = open;
                        // calculate fields
                        double priorQty = order.getExecuted();
                        double priorAvg = order.getAvgPx();
                        if (random.nextBoolean())
                            fillPrice += 0.01;
                        else
                            fillPrice -= 0.01;
                        double thisAvg = ((fillQty*fillPrice)
                                + (priorQty*priorAvg))
                                / (priorQty + fillQty);
                        int pricePrecision = 4;
                        try {
                            pricePrecision =

(int)settings.getLong("FIXimulatorPricePrecision");
                        } catch ( Exception e ) {}
                        double factor = Math.pow( 10, pricePrecision );
                        fillPrice = Math.round( fillPrice * factor ) /
factor;
                        thisAvg = Math.round( thisAvg * factor ) /
factor;
                        //update order
                        order.setOpen( open - fillQty );
                        order.setStatus(OrdStatus.FILLED);
                        order.setExecuted(order.getExecuted() +
fillQty);
```

```
                        order.setAvgPx(thisAvg);
                        orders.update();
                        // create execution
                        Execution partial = new Execution(order);
                        partial.setExecType(ExecType.FILL);
                        partial.setExecTranType(ExecTransType.NEW);
                        partial.setLeavesQty(order.getOpen());
                        partial.setCumQty(order.getQuantity()-
order.getOpen());
                        partial.setAvgPx(thisAvg);
                        partial.setLastShares(fillQty);
                        partial.setLastPx(fillPrice);
                        sendExecution(partial);
                        break;
                    }
                }
                try {
                    Thread.sleep( delay.longValue() );
                } catch ( InterruptedException e ) {}
            }
        }
    }
}
```

IOI.java

```
/*
 * File     : IOI.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is a basic IOI object that is used to create
 *            and store ioi details.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.core;

public class IOI implements Cloneable {
    private static int nextID = 1;
    private String ID = null;
    private String refID = null;
    private String type = null;     // NEW, CANCEL, REPLACE
    private String side = "";       // BUY, SELL, UNDISCLOSED
    private Integer quantity = 0;
    private String symbol = "";
    private String securityID = "";
    private String iDSource = "";   //
    private double price = 0.0;
    private String natural = "";    // YES, NO

    public IOI () {
        ID = generateID();
    }

    @Override
```

```
public IOI clone() {
    try {
        IOI ioi = (IOI)super.clone();
        ioi.setRefID(getID());
        ioi.setID(ioi.generateID());
        return ioi;
    } catch(CloneNotSupportedException e) {}
    return null;
}

public String generateID() {
    return "I" + Long.valueOf(
            System.currentTimeMillis()+(nextID++)).toString();
}

public String getID() {
    return ID;
}

public void setID(String id) {
    this.ID = id;
}

public String getRefID() {
    return refID;
}

public void setRefID(String refID) {
    this.refID = refID;
}

public String getType() {
    return type;
}

public void setType(String type) {
    this.type = "NEW";
    if (type.toUpperCase().startsWith("C"))
        this.type = "CANCEL";
    if (type.toUpperCase().startsWith("R"))
        this.type = "REPLACE";
}

public String getNatural() {
    return natural;
}

public void setNatural(String natural) {
    this.natural = "NO";
    if (natural.toUpperCase().startsWith("Y"))
        this.natural = "YES";
}

public double getPrice() {
    return price;
}
```

```
    public void setPrice(double price) {
        this.price = price;
    }

    public Integer getQuantity() {
        return quantity;
    }

    public void setQuantity(Integer quantity) {
        this.quantity = quantity;
    }

    public String getSecurityID() {
        return securityID;
    }

    public void setSecurityID(String securityID) {
        if (securityID.equals(""))
            this.securityID = "<MISSING>";
        else
            this.securityID = securityID;
    }

    public String getIDSource() {
        return iDSource;
    }

    public void setIDSource(String iDSource) {
        this.iDSource = "UNKNOWN";
        if (iDSource.toUpperCase().startsWith("C")) this.iDSource =
"CUSIP";
        if (iDSource.toUpperCase().startsWith("S")) this.iDSource =
"SEDOL";
        if (iDSource.toUpperCase().startsWith("T")) this.iDSource =
"TICKER";
        if (iDSource.toUpperCase().startsWith("R")) this.iDSource =
"RIC";
    }

    public String getSide() {
        return side;
    }

    public void setSide(String side) {
        this.side = "BUY";
        if (side.toUpperCase().startsWith("S")) this.side = "SELL";
        if (side.toUpperCase().startsWith("U")) this.side =
"UNDISCLOSED";
    }

    public String getSymbol() {
        return symbol;
    }

    public void setSymbol(String symbol) {
        if (symbol.equals(""))
            this.symbol = "<MISSING>";
        else
            this.symbol = symbol;
    }

}
```

IOIset.java

```
/*
 * File     : IOIset.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is a Set of IOI objects with a utility
 *            methods to access the individual iois.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.core;

import java.util.ArrayList;
import java.util.Iterator;
import edu.harvard.fas.zfeledy.fiximulator.ui.IOITableModel;

public class IOIset {
    private ArrayList<IOI> iois = new ArrayList<IOI>();
    private IOITableModel ioiTableModel = null;

    public IOIset() {}

    public void add ( IOI ioi ) {
        iois.add( ioi );
        int limit = 50;
        try {
            limit = (int)FIXimulator.getApplication().getSettings()
                    .getLong("FIXimulatorCachedObjects");
        } catch ( Exception e ) {}
        while ( iois.size() > limit ) {
            iois.remove(0);
        }
        ioiTableModel.update();
    }

    public void addCallback(IOITableModel ioiTableModel){
        this.ioiTableModel = ioiTableModel;
    }

    public int getCount() {
        return iois.size();
    }

    public IOI getIOI( int i ) {
        return iois.get( i );
    }

    public IOI getIOI( String id ) {
        Iterator<IOI> iterator = iois.iterator();
        while ( iterator.hasNext() ){
            IOI ioi = iterator.next();
            if ( ioi.getID().equals(id) )
                return ioi;
        }
        return null;
    }
}
```

Instrument.java

```
/*
 * File     : Instrument.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is a basic Instrument object that is used to
 *            create and store instrument details.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.core;

public class Instrument {

      private String ticker;
      private String cusip;
      private String sedol;
      private String name;
      private String ric;
      private String price;

      public Instrument( String ticker, String sedol, String name,
                  String ric, String cusip, String price ) {
            this.ticker = ticker;
            this.cusip = cusip;
            this.sedol = sedol;
            this.name = name;
            this.ric = ric;
            this.price = price;
      }

      public String getTicker() {
            return ticker;
      }

      public String getCusip() {
            return cusip;
      }

      public String getSedol() {
            return sedol;
      }

      public String getName() {
            return name;
      }

      public String getRIC() {
            return ric;
      }

      public String getPrice() {
            return price;
      }
}
```

InstrumentSet.java

```
/*
 * File     : InstrumentSet.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is a Set of Instrument objects with a utility
 *            methods to to access the individual instruments.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.core;

import java.io.BufferedInputStream;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import edu.harvard.fas.zfeledy.fiximulator.ui.InstrumentTableModel;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;


public class InstrumentSet extends DefaultHandler {
    private ArrayList<Instrument> instruments = new
ArrayList<Instrument>();
    private ArrayList<Instrument> oldInstruments = new
ArrayList<Instrument>();
    private InstrumentTableModel instrumentModel = null;

    public InstrumentSet( File file ) {
        try {
            InputStream input =
                    new BufferedInputStream(new FileInputStream(file));
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( input, this);
        } catch ( Exception e ) {
            System.out.println( "Error reading/parsing instrument
file." );
            e.printStackTrace();
        }
    }

    public void reloadInstrumentSet( File file ) {
        try {
            oldInstruments.clear();
            oldInstruments.addAll(instruments);
            instruments.clear();
            InputStream input =
                    new BufferedInputStream(new FileInputStream(file));
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( input, this);
            instrumentModel.update();
        } catch ( Exception e ) {
            System.out.println( "Error reading/parsing instrument
file." );
            e.printStackTrace();
            instruments.clear();
            instruments.addAll(oldInstruments);
        }
    }

    @Override
    public void startElement( String namespace, String localName,
                    String qualifiedName, Attributes attributes ){
        if ( qualifiedName.equals( "instrument" )) {
            String ticker = attributes.getValue( "ticker" );
            String cusip = attributes.getValue( "cusip" );
            String sedol = attributes.getValue( "sedol" );
            String name = attributes.getValue( "name" );
            String ric = attributes.getValue( "ric" );
            String price = attributes.getValue( "price" );
            Instrument instrument =
                    new Instrument( ticker, sedol, name, ric, cusip,
price );
            instruments.add( instrument );
        }
    }

    public int getCount() {
        return instruments.size();
    }

    public Instrument getInstrument( int i ) {
        return instruments.get( i );
    }

    public Instrument getInstrument( String identifier ) {
        Iterator<Instrument> iterator = instruments.iterator();
        while ( iterator.hasNext() ){
            Instrument instrument = iterator.next();
            if ( instrument.getTicker().equals( identifier ) ||
                 instrument.getSedol().equals( identifier ) ||
                 instrument.getCusip().equals( identifier ) ||
                 instrument.getName().equals( identifier ))
                return instrument;
        }
        return null;
    }

    public Instrument randomInstrument() {
        Instrument instrument = null;
        Random generator = new Random();
        int size = instruments.size();
        int index = generator.nextInt( size );
        instrument = instruments.get( index );
        return instrument;
    }

    public void outputToXML() {
        try {
            BufferedWriter writer =
                    new BufferedWriter(new
FileWriter("config/instruments.xml"));
            writer.write("<?xml version=\"1.0\" encoding=\"UTF-
8\"?>\n");
            writer.write("<instruments>\n");
            Iterator<Instrument> iterator = instruments.iterator();
            while (iterator.hasNext()) {
                Instrument instrument = (Instrument)iterator.next();
                String output = "   <instrument";
                output += " name=\"" + instrument.getName() + "\"";
                output += " ticker=\"" + instrument.getTicker() + "\"";
                output += " cusip=\"" + instrument.getCusip() + "\"";
                output += " sedol=\"" + instrument.getSedol() + "\"";
                output += " ric=\"" + instrument.getRIC() + "\"";
                output += " price=\"" + instrument.getPrice() + "\"";
                output += "/>\n";
                writer.write( output );
            }
            writer.write("</instruments>\n");
            writer.close();
        } catch ( IOException e ) {e.printStackTrace();}
    }

    public void addCallback(InstrumentTableModel instrumentModel){
        this.instrumentModel = instrumentModel;
    }
}
```

LogMessage.java

```
/*
 * File     : LogMessage.java
 *
 * Author   : Brian M. Coyner
 *
 * Contents : This class is a basic LogMessage object that is used to
 *            create and store log message details.  The file was
 *            taken from the Log4FIX project and adapted for the needs
 *            of FIXimulator by Zoltan Feledy.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.core;
/*
 * The Log4FIX Software License
 * Copyright (c) 2006 - 2007 opentradingsolutions.org  All rights
reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. Neither the name of the product (Log4FIX), nor
opentradingsolutions.org,
 *    nor the names of its contributors may be used to endorse or
promote
 *    products derived from this software without specific prior
written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL OPENTRADINGSOLUTIONS.ORG OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import edu.harvard.fas.zfeledy.fiximulator.util.FIXMessageHelper;
import edu.harvard.fas.zfeledy.fiximulator.util.LogField;
import edu.harvard.fas.zfeledy.fiximulator.util.LogGroup;

import quickfix.DataDictionary;
import quickfix.Field;
import quickfix.FieldConvertError;
import quickfix.FieldNotFound;
import quickfix.Group;
import quickfix.InvalidMessage;
import quickfix.Message;
import quickfix.SessionID;
import quickfix.field.MsgType;

/**
 * @author Brian M. Coyner
 */
public class LogMessage implements Comparable<Object> {

    public static final char DEFAULT_DELIMETER = '|';
    public static final char SOH_DELIMETER = (char) 0x01;

    public static final String INCOMING = "incoming";
    public static final String MESSAGE_TYPE_NAME = "messageTypeName";
    public static final String SENDING_TIME = "sendingTime";
    public static final String RAW_MESSAGE = "rawMessage";

    private SessionID sessionId;
    private boolean incoming;
    private String rawMessage;
    private String messageTypeName;
    private Date sendingTime;
    private DataDictionary dictionary;
    //private List<ValidationError> validationErrors;
    private boolean isValid;
    private int messageIndex;

    public LogMessage(int messageIndex, boolean incoming, SessionID 
sessionId,
            String rawMessage, DataDictionary dictionary) {
        this.messageIndex = messageIndex;

        isValid = true;
        this.dictionary = dictionary;
        this.rawMessage = rawMessage.replace(SOH_DELIMETER, 
DEFAULT_DELIMETER);
        this.sessionId = sessionId;
        this.incoming = incoming;

        sendingTime = lookupSendingTime();
        messageTypeName = lookupMessageTypeName();
    }

    public SessionID getSessionId() {
        return sessionId;
    }

    public String getRawMessage() {
        return rawMessage;
    }

    public String getMessageTypeName() {
        return messageTypeName;
    }

    public int getMessageIndex() {
      return messageIndex;
    }

    /**
     * A message is valid if all required fields are found in the
message and
     * the checksum value is correct. If true then the
     * {@link #getValidationErrorMessages()} has one or more
     * <code>Exception</code>s explaining the problems with the message
     * as reported by QuickFIX/J.
     *
     * @return true if the message is valid.
     */
    public boolean isValid() {
        return isValid;
    }

    public boolean isIncoming() {
        return incoming;
    }

    /**
     * The sending time of the message. This value may be null if the
     * {@link SendingTime} field is not found in the message. If the
value is
     * null then the {@link #getValidationErrorMessages()} contains an
     * <code>Exception</code> describing the problem.
     * @return the sending time of the message or null if the message
was
     * missing the sending time.
     */
    public Date getSendingTime() {
        return sendingTime;
    }

    /**
     * This method collects all <code>Fields</code> into a map. Each
field
     * is picked out of the raw message string and is looked up from
the map.
     * This ensures that we display the fields in the same order as the
raw
     * message string. Why? Because the message class does not provide
a way to
     * get the fields in the sent order. The only time we care about
the field
     * order is when logging.
     * This method executes on the Event Dispatch Thread, so we are not
slowing
```

```
     * down the quickfix thread.
     *
     * This object does <strong>not</strong> cache the field objects.
Each
     * invocation of this method creates a new list of
<code>LogField</code>
     * objects. This is done so that memory utilization is kept low.
The caller
     * should clear the returned list when it is finished with the
values. The
     * typical caller would clear the list when a new message is
displayed.
     * In fact, this is exactly what the
     * {@link org.opentradingsolutions.log4fix.ui.messages.ViewModel}
does.
     * @return locally created <tt>List</tt> of <tt>LogField</tt>
objects;
     * not a cached value.
     */
    public List<LogField> getLogFields() {


        Message message = createMessage();

        List<LogField> logFields = new ArrayList<LogField>();

        Map<Integer, Field> allFields = getAllFields(message);

        String[] fields = rawMessage.split("\\|");

        for (String fieldString : fields) {
            int indexOfEqual = fieldString.indexOf('=');
            int tag = Integer.parseInt(fieldString.substring(0,
indexOfEqual));

            Field field = allFields.remove(tag);
            if (field != null) {
                logFields.add(createLogField(message, field));
            }
        }

        return logFields;
    }

    public int compareTo(Object o) {
        LogMessage rhs = (LogMessage) o;
        int rhsMessageIndex = rhs.messageIndex;
        return (messageIndex < rhsMessageIndex ? -1 :
                (messageIndex == rhsMessageIndex ? 0 : 1));
    }


    @Override
    public String toString() {
        return "" + messageIndex;
    }
    @SuppressWarnings("unchecked")
      private LogField createLogField(Message message, Field field) {

        MsgType messageType = getMessageType(message);
        String messageTypeValue = messageType.getValue();

        LogField logField =
                LogField.createLogField(messageType, field,
dictionary);

        final DataDictionary.GroupInfo groupInfo = dictionary.getGroup(
                    messageTypeValue, field.getTag());
        if (groupInfo != null) {

            int delimeterField = groupInfo.getDelimeterField();
            Group group = new Group(field.getTag(), delimeterField);
            int numberOfGroups =  Integer.valueOf((String)
field.getObject());
            for (int index = 0; index < numberOfGroups; index++) {
                LogGroup logGroup =
                        new LogGroup(messageType, field, dictionary);

                try {

                    message.getGroup(index + 1, group);

                    Iterator groupIterator = group.iterator();
                    while (groupIterator.hasNext()) {
                        Field groupField = (Field)
groupIterator.next();

logGroup.addField(LogField.createLogField(messageType,
                                groupField, dictionary));

                    }
                } catch (FieldNotFound fieldNotFound) {
                }

                logField.addGroup(logGroup);
            }
        }

        return logField;
    }

    private Message createMessage() {
        String sohMessage =
                rawMessage.replace(DEFAULT_DELIMETER, SOH_DELIMETER);
        try {
            return new Message(sohMessage, dictionary, true);
        } catch (InvalidMessage invalidMessage) {
            try {
                return new Message(sohMessage, dictionary, false);
            } catch (InvalidMessage ugh) {
                return null;
            }
        }
    }

    @SuppressWarnings("unchecked")
      private Map<Integer, Field> getAllFields(Message genericMessage)
{
        Map<Integer, Field> allFields = new LinkedHashMap<Integer,
Field>();

        Iterator iterator = genericMessage.getHeader().iterator();
        while (iterator.hasNext()) {
            Field field = (Field) iterator.next();
            allFields.put(field.getTag(), field);
        }

        iterator = genericMessage.iterator();
        while (iterator.hasNext()) {
            Field field = (Field) iterator.next();
            int tag = field.getTag();
            if (!allFields.containsKey(tag)) {
                allFields.put(tag, field);
            }
        }

        iterator = genericMessage.getTrailer().iterator();
        while (iterator.hasNext()) {
            Field field = (Field) iterator.next();
            allFields.put(field.getTag(), field);
        }

        return allFields;
    }

    private String lookupMessageTypeName() {
        String messageTypeValue =
FIXMessageHelper.getMessageType(rawMessage,
                DEFAULT_DELIMETER);
        if (messageTypeValue == null) {
            isValid = false;
            return null;
        }
        return dictionary.getValueName(MsgType.FIELD,
messageTypeValue);
    }

    private Date lookupSendingTime() {
        try {
            Date date = FIXMessageHelper.getSendingTime(
                    rawMessage, DEFAULT_DELIMETER);
            if (date == null) {
                return date;
            }
            return date;
        } catch (FieldConvertError fieldConvertError) {
            return null;
        }
    }
    private MsgType getMessageType(Message message) {
        try {
            return (MsgType) message.getHeader().getField(new
MsgType());
        } catch (FieldNotFound fieldNotFound) {
            throw new RuntimeException(fieldNotFound);
        }
    }
}
```

LogMessageSet.java

```
/*
 * File     : LogMessageSet.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is a Set of LogMessage objects with utility
 *            methods to access the individual messages.
 */

package edu.harvard.fas.zfeledy.fiximulator.core;

import java.util.ArrayList;
import edu.harvard.fas.zfeledy.fiximulator.ui.MessageTableModel;
import quickfix.DataDictionary;
import quickfix.Message;
import quickfix.SessionID;


public class LogMessageSet {
      private static final long serialVersionUID = 1L;
      private ArrayList<LogMessage> messages = null;
      private MessageTableModel model;
      private int messageIndex = 0;

      public LogMessageSet() {
            messages = new ArrayList<LogMessage>();
      }

      public void add(Message message, boolean incoming,
                DataDictionary dictionary, SessionID sessionID) {
            messageIndex++;
            LogMessage msg =
                        new LogMessage(messageIndex, incoming,
sessionID,
                        message.toString(), dictionary);
                messages.add(msg);
                int limit = 50;
                try {
                    limit =
(int)FIXimulator.getApplication().getSettings()
                            .getLong("FIXimulatorCachedObjects");
                } catch ( Exception e ) {}
                while ( messages.size() > limit ) {
                    messages.remove(0);
            }
        //call back to the model to update
        model.update();
    }

    public LogMessage getMessage( int i ) {
        return messages.get(i);
    }

    public int getCount(){
        return messages.size();
    }

    public void addCallback(MessageTableModel model) {
        this.model = model;
    }
}
```

Order.java

```
/*
 * File     : Order.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is a basic Order object that is used to
 *            create and store order details.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.core;

import quickfix.FieldNotFound;
import quickfix.field.ClOrdID;
import quickfix.field.OrdType;
import quickfix.field.OrderQty;
import quickfix.field.OrigClOrdID;
import quickfix.field.Price;
import quickfix.field.SecurityID;
import quickfix.field.IDSource;
import quickfix.field.Side;
import quickfix.field.Symbol;
import quickfix.field.TimeInForce;

public class Order implements Cloneable {
    private static int nextID = 1;
    private boolean receivedOrder = false;
    private boolean receivedCancel = false;
    private boolean receivedReplace = false;
    private boolean rejectedCancelReplace = false;
    private char side;
    private char type;
    private char tif = '0';     // Day order if omitted
    private char status;
    private String ID = null;
    private String clientID = null;
    private String origClientID = null;
    private String symbol = null;
    private String securityID = null;
    private String idSource = null;
    private double quantity = 0.0;
    private double open = 0.0;
    private double executed = 0.0;
    private double limit = 0.0;
    private double avgPx = 0.0;

    @Override
    public Order clone() {
        try {
            Order order = (Order)super.clone();
            order.setOrigClientID(getID());
            order.setID(generateID());
            return order;
        } catch ( CloneNotSupportedException e ) {}
        return null;
    }

    public Order () {
        ID = generateID();
    }

    public Order( quickfix.fix42.NewOrderSingle message ) {
        ID = generateID();

        // ClOrdID
        try {
            ClOrdID clOrdID = new ClOrdID();
            message.get(clOrdID);
            this.setClientID(clOrdID.getValue().toString());
        } catch (FieldNotFound ex) {}

        // Side
        try {
            Side msgSide = new Side();
            message.get(msgSide);
            setSide(msgSide.getValue());
        } catch (FieldNotFound ex) {}

        // Symbol
        try {
            Symbol msgSymbol = new Symbol();
            message.get(msgSymbol);
            setSymbol(msgSymbol.getValue());
        } catch (FieldNotFound ex) {}

        // Type
        try {
            OrdType msgType = new OrdType();
            message.get(msgType);
            setType(msgType.getValue());
        } catch (FieldNotFound ex) {}

        // OrderQty
```

```
        try {
            OrderQty msgQty = new OrderQty();
            message.get(msgQty);
            setQuantity(msgQty.getValue());
            setOpen(msgQty.getValue());
        } catch (FieldNotFound ex) {}

        // TimeInForce
        try {
            TimeInForce msgTIF = new TimeInForce();
            message.get(msgTIF);
            this.setTif(msgTIF.getValue());
        } catch (FieldNotFound ex) {}

        // Price
        try {
            Price price = new Price();
            message.get(price);
            this.setLimit(price.getValue());
        } catch (FieldNotFound ex) {}

        // SecurityID
        try {
            SecurityID secID = new SecurityID();
            message.get(secID);
            this.setSecurityID(secID.getValue());
        } catch (FieldNotFound ex) {}

        // IDSource
        try {
            IDSource idSrc = new IDSource();
            message.get(idSrc);
            this.setIdSource(idSrc.getValue());
        } catch (FieldNotFound ex) {}

    }

    public Order( quickfix.fix42.OrderCancelRequest message ) {
        ID = generateID();

        // ClOrdID
        try {
            ClOrdID clOrdID = new ClOrdID();
            message.get(clOrdID);
            this.setClientID(clOrdID.getValue().toString());
        } catch (FieldNotFound ex) {}

        // OrigClOrdID
        try {
            OrigClOrdID origClOrdID = new OrigClOrdID();
            message.get(origClOrdID);
            this.setOrigClientID(origClOrdID.getValue().toString());
        } catch (FieldNotFound ex) {}

        Order oldOrder = FIXimulator.getApplication()
                .getOrders().getOrder(origClientID);
        if ( oldOrder != null ) {
            open = oldOrder.getOpen();
            executed = oldOrder.getExecuted();
            limit = oldOrder.getLimit();
            avgPx = oldOrder.getAvgPx();
            status = oldOrder.getFIXStatus();
        }

        // Side
        try {
            Side msgSide = new Side();
            message.get(msgSide);
            setSide(msgSide.getValue());
        } catch (FieldNotFound ex) {}

        // Symbol
        try {
            Symbol msgSymbol = new Symbol();
            message.get(msgSymbol);
            setSymbol(msgSymbol.getValue());
        } catch (FieldNotFound ex) {}

        // OrderQty
        try {
            OrderQty msgQty = new OrderQty();
            message.get(msgQty);
            setQuantity(msgQty.getValue());
            setOpen(msgQty.getValue());
        } catch (FieldNotFound ex) {}

        // SecurityID
        try {
            SecurityID secID = new SecurityID();
            message.get(secID);
            this.setSecurityID(secID.getValue());
        } catch (FieldNotFound ex) {}

        // IDSource
        try {
            IDSource idSrc = new IDSource();
            message.get(idSrc);
            this.setIdSource(idSrc.getValue());
        } catch (FieldNotFound ex) {}
    }

    public Order( quickfix.fix42.OrderCancelReplaceRequest message ) {
        ID = generateID();

        // ClOrdID
        try {
            ClOrdID clOrdID = new ClOrdID();
            message.get(clOrdID);
            this.setClientID(clOrdID.getValue().toString());
        } catch (FieldNotFound ex) {}

        // OrigClOrdID
        try {
            OrigClOrdID origClOrdID = new OrigClOrdID();
            message.get(origClOrdID);
            this.setOrigClientID(origClOrdID.getValue().toString());
        } catch (FieldNotFound ex) {}

        Order oldOrder = FIXimulator.getApplication()
                .getOrders().getOrder(origClientID);
        if ( oldOrder != null ) {
            open = oldOrder.getOpen();
            executed = oldOrder.getExecuted();
            avgPx = oldOrder.getAvgPx();
            status = oldOrder.getFIXStatus();
        }

        // Side
        try {
            Side msgSide = new Side();
            message.get(msgSide);
            setSide(msgSide.getValue());
        } catch (FieldNotFound ex) {}

        // Symbol
        try {
            Symbol msgSymbol = new Symbol();
            message.get(msgSymbol);
            setSymbol(msgSymbol.getValue());
        } catch (FieldNotFound ex) {}

        // Type
        try {
            OrdType msgType = new OrdType();
            message.get(msgType);
            setType(msgType.getValue());
        } catch (FieldNotFound ex) {}

        // OrderQty
        try {
            OrderQty msgQty = new OrderQty();
            message.get(msgQty);
            setQuantity(msgQty.getValue());
            setOpen(msgQty.getValue());
        } catch (FieldNotFound ex) {}

        // TimeInForce
        try {
            TimeInForce msgTIF = new TimeInForce();
            message.get(msgTIF);
            this.setTif(msgTIF.getValue());
        } catch (FieldNotFound ex) {}

        // Price
        try {
            Price price = new Price();
            message.get(price);
            this.setLimit(price.getValue());
        } catch (FieldNotFound ex) {}

        // SecurityID
```

```
        try {
            SecurityID secID = new SecurityID();
            message.get(secID);
            this.setSecurityID(secID.getValue());
        } catch (FieldNotFound ex) {}

        // IDSource
        try {
            IDSource idSrc = new IDSource();
            message.get(idSrc);
            this.setIdSource(idSrc.getValue());
        } catch (FieldNotFound ex) {}
    }

    public String generateID() {
        return "O" + Long.valueOf(
                System.currentTimeMillis()+(nextID++)).toString();
    }

    public String getID() {
        return ID;
    }

    public void setID(String ID) {
        this.ID = ID;
    }

    public String getClientID() {
        return clientID;
    }

    public void setClientID(String clientID) {
        this.clientID = clientID;
    }

    public String getOrigClientID() {
        return origClientID;
    }

    public void setOrigClientID(String origClientID) {
        this.origClientID = origClientID;
    }

    public double getLimit() {
        return limit;
    }

    public void setLimit(double limit) {
        this.limit = limit;
    }

    public double getAvgPx() {
        return avgPx;
    }

    public void setAvgPx(double avgPx) {
        this.avgPx = avgPx;
    }

    public double getExecuted() {
        return executed;
    }

    public void setExecuted(double executed) {
        this.executed = executed;
    }

    public static int getNextID() {
        return nextID;
    }

    public static void setNextID(int nextID) {
        Order.nextID = nextID;
    }

    public double getOpen() {
        return open;
    }

    public void setOpen(double open) {
        this.open = open;
    }

    public double getQuantity() {
        return quantity;
    }

    public void setQuantity(double quantity) {
        this.quantity = quantity;
    }

    public String getSide() {
        if (side == '1') return "Buy";
        if (side == '2') return "Sell";
        if (side == '3') return "Buy minus";
        if (side == '4') return "Sell plus";
        if (side == '5') return "Sell short";
        if (side == '6') return "Sell short exempt";
        if (side == '7') return "Undisclosed";
        if (side == '8') return "Cross";
        if (side == '9') return "Cross short";
        return "<UNKNOWN>";
    }

    public char getFIXSide() {
        return side;
    }

    public void setSide(char side) {
        this.side = side;
    }

    public String getStatus() {
        if (receivedOrder) return "Received";
        if (receivedCancel) return "Cancel Received";
        if (receivedReplace) return "Replace Received";
        if (rejectedCancelReplace) return "Cancel/Replace Rejected";
        if (status == '0') return "New";
        if (status == '1') return "Partially filled";
        if (status == '2') return "Filled";
        if (status == '3') return "Done for day";
        if (status == '4') return "Canceled";
        if (status == '5') return "Replaced";
        if (status == '6') return "Pending Cancel";
        if (status == '7') return "Stopped";
        if (status == '8') return "Rejected";
        if (status == '9') return "Suspended";
        if (status == 'A') return "Pending New";
        if (status == 'B') return "Calculated";
        if (status == 'C') return "Expired";
        if (status == 'D') return "Accepted for bidding";
        if (status == 'E') return "Pending Replace";
        return "<UNKNOWN>";
     }

    public char getFIXStatus() {
        return status;
    }

    public void setStatus(char status) {
        this.status = status;
    }

    public String getSymbol() {
        return symbol;
    }

    public void setSymbol(String symbol) {
        this.symbol = symbol;
    }

    public String getTif() {
        if (tif == '0') return "Day";
        if (tif == '1') return "GTC";
        if (tif == '2') return "OPG";
        if (tif == '3') return "IOC";
        if (tif == '4') return "FOK";
        if (tif == '5') return "GTX";
        if (tif == '6') return "GTD";
        return "<UNKNOWN>";
    }

    public char getFIXTif() {
        return tif;
    }

    public void setTif(char tif) {
        this.tif = tif;
    }

    public String getIdSource() {
        return idSource;
    }

    public void setIdSource(String idSource) {
        this.idSource = idSource;
    }

    public String getSecurityID() {
        return securityID;
    }

    public void setSecurityID(String securityID) {
        this.securityID = securityID;
    }

    public String getType() {
        if (type == '1') return "Market";
        if (type == '2') return "Limit";
        if (type == '3') return "Stop";
        if (type == '4') return "Stop limit";
        if (type == '5') return "Market on close";
        if (type == '6') return "With or without";
        if (type == '7') return "Limit or better";
        if (type == '8') return "Limit with or without";
        if (type == '9') return "On basis";
        if (type == 'A') return "On close";
        if (type == 'B') return "Limit on close";
        if (type == 'C') return "Forex - Market";
        if (type == 'D') return "Previously quoted";
        if (type == 'E') return "Previously indicated";
        if (type == 'F') return "Forex - Limit";
        if (type == 'G') return "Forex - Swap";
        if (type == 'H') return "Forex - Previously Quoted";
        if (type == 'I') return "Funari";
        if (type == 'P') return "Pegged";
        return "<UNKNOWN>";
    }

    public char getFIXType() {
        return type;
    }

    public void setType(char type) {
        this.type = type;
    }

    public boolean isReceivedCancel() {
        return receivedCancel;
    }

    public void setReceivedCancel(boolean receivedCancel) {
        this.receivedCancel = receivedCancel;
    }

    public boolean isReceivedOrder() {
        return receivedOrder;
    }
    public void setReceivedOrder(boolean receivedOrder) {
        this.receivedOrder = receivedOrder;
    }

    public boolean isReceivedReplace() {
        return receivedReplace;
    }

    public void setReceivedReplace(boolean receivedReplace) {
        this.receivedReplace = receivedReplace;
    }

    public boolean isRejectedCancelReplace() {
        return rejectedCancelReplace;
    }

    public void setRejectedCancelReplace(boolean rejectedCancelReplace)
{
        this.rejectedCancelReplace = rejectedCancelReplace;
    }
}
```

OrderSet.java

```
/*
 * File     : OrderSet.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is a Set of Order objects with a utility
 *            methods to access the individual orders.
 */

package edu.harvard.fas.zfeledy.fiximulator.core;

import java.util.ArrayList;
import java.util.Iterator;
import edu.harvard.fas.zfeledy.fiximulator.ui.OrderTableModel;

public class OrderSet {
    private ArrayList<Order> orders = new ArrayList<Order>();
    private ArrayList<Order> ordersToFill = new ArrayList<Order>();
    private OrderTableModel orderTableModel = null;

    public OrderSet() {}

    public void add ( Order order, boolean toFill ) {
        orders.add( order );
        if ( toFill ) ordersToFill.add(order);
        int limit = 50;
        try {
            limit = (int)FIXimulator.getApplication().getSettings()
                     .getLong("FIXimulatorCachedObjects");
        } catch ( Exception e ) {}
        while ( orders.size() > limit ) {
            orders.remove(0);
        }
        orderTableModel.update();
    }

    public void update () {
        orderTableModel.update();
    }

    public void addCallback(OrderTableModel orderTableModel){
        this.orderTableModel = orderTableModel;
    }

    public int getCount() {
        return orders.size();
    }

    public Order getOrder( int i ) {
        return orders.get( i );
    }

    public Order getOrder( String id ) {
        Iterator<Order> iterator = orders.iterator();
        while ( iterator.hasNext() ){
            Order order = iterator.next();
            if ( order.getID().equals(id) ||
order.getClientID().equals(id))
                return order;
        }
        return null;
    }

    public boolean haveOrdersToFill() {
        if ( ordersToFill.size() > 0 ) return true;
        return false;
    }

    public Order getOrderToFill() {
        return ordersToFill.remove(0);
    }
}
```

ExecutionCellRenderer.java

```
/*
 * File     : ExecutionCellRenderer.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This renderer is used on the JTable that displays
 *            Executions and colors the executions that have received
 *            a DontKnowTrade.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.ui;
```

```
import java.awt.Color;
import java.awt.Component;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;

public class ExecutionCellRenderer  extends DefaultTableCellRenderer {

    @Override
    public Component getTableCellRendererComponent(JTable table, Object
value,
            boolean isSelected, boolean hasFocus, int row, int column)
{

        int myRow = table.convertRowIndexToModel(row);
        Component component =
super.getTableCellRendererComponent(table, value,
                                          isSelected, hasFocus, myRow,
column);
        Boolean DKd = (Boolean)((ExecutionTableModel)table.getModel())
                .getValueAt(myRow, 12);

        if ( DKd ) {
            component.setForeground(Color.RED);
        }
        if ( !DKd ) {
            component.setForeground(Color.BLACK);
        }
        return component;
    }
}
```

ExecutionTableModel.java

```
/*
 * File     : ExecutionTableModel.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is the TableModel for the Execution Table.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.ui;

import javax.swing.table.AbstractTableModel;
import edu.harvard.fas.zfeledy.fiximulator.core.FIXimulator;
import edu.harvard.fas.zfeledy.fiximulator.core.Execution;
import edu.harvard.fas.zfeledy.fiximulator.core.ExecutionSet;
import edu.harvard.fas.zfeledy.fiximulator.core.Order;

public class ExecutionTableModel extends AbstractTableModel {
    private static ExecutionSet executions =
            FIXimulator.getApplication().getExecutions();
    private static String[] columns =
        {"ID", "ClOrdID", "Side", "Symbol", "LastQty", "LastPx",
          "CumQty", "AvgPx", "Open", "ExecType", "ExecTranType",
"RefID", "DKd"};


    public ExecutionTableModel(){
        FIXimulator.getApplication().getExecutions().addCallback(this);
    }

    public int getColumnCount() {
        return columns.length;
    }

    @Override
    public String getColumnName(int column) {
        return columns[column];
    }

    @Override
    public Class getColumnClass(int column) {
        if (column == 4) return Double.class;
        if (column == 5) return Double.class;
        if (column == 6) return Double.class;
        if (column == 7) return Double.class;
        if (column == 8) return Double.class;
        return String.class;
    }

    public int getRowCount() {
        return executions.getCount();
    }

    public Object getValueAt(int row, int column) {
        Execution execution = executions.getExecution(row);
        Order order = execution.getOrder();
        if (column == 0) return execution.getID();
        if (column == 1) return order.getClientID();
        if (column == 2) return order.getSide();
        if (column == 3) return order.getSymbol();
        if (column == 4) return execution.getLastShares();
        if (column == 5) return execution.getLastPx();
        if (column == 6) return execution.getCumQty();
        if (column == 7) return execution.getAvgPx();
        if (column == 8) return order.getOpen();
        if (column == 9) return execution.getExecType();
        if (column == 10) return execution.getExecTranType();
        if (column == 11) return execution.getRefID();
        if (column == 12) return execution.isDKd();
        return "";
    }

    public void update() {
        fireTableDataChanged();
    }
}
```

FIXimulatorFrame.java

```
/*
 * File     : FIXimulatorFrame.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This is the main application class.  It is combined with
 *            the user interface code most of which was autogenerated
 *            by NetBeans.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.ui;

import edu.harvard.fas.zfeledy.fiximulator.core.Execution;
import java.io.File;
import javax.swing.UIManager;
import edu.harvard.fas.zfeledy.fiximulator.core.FIXimulator;
import edu.harvard.fas.zfeledy.fiximulator.core.IOI;
import edu.harvard.fas.zfeledy.fiximulator.core.Order;

public class FIXimulatorFrame extends javax.swing.JFrame {
    private static FIXimulator fiximulator;
    private IOI dialogIOI = null;
    private Execution dialogExecution = null;

    /** Creates new form FIXimulatorFrame */
    public FIXimulatorFrame() {
        try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {e.printStackTrace();}
        initComponents();
    }

    public IOI getDialogIOI() {
        return dialogIOI;
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        bindingGroup = new org.jdesktop.beansbinding.BindingGroup();

        aboutDialog = new javax.swing.JDialog();
        aboutPanel = new javax.swing.JPanel();
        okButton = new javax.swing.JButton();
        aboutDialogLabel = new javax.swing.JLabel();
        instrumentFileChooser = new javax.swing.JFileChooser();
        ioiDialog = new javax.swing.JDialog();
```

```
ioiDialogOK = new javax.swing.JButton();
ioiDialogCancel = new javax.swing.JButton();
ioiIDLabel = new javax.swing.JLabel();
ioiSideLabel = new javax.swing.JLabel();
ioiSharesLabel = new javax.swing.JLabel();
ioiSymbolLabel = new javax.swing.JLabel();
ioiSecurityIDLabel = new javax.swing.JLabel();
ioiIDSourceLabel = new javax.swing.JLabel();
ioiPriceLabel = new javax.swing.JLabel();
ioiNaturalLabel = new javax.swing.JLabel();
ioiDialogID = new javax.swing.JLabel();
ioiDialogSide = new javax.swing.JComboBox();
ioiDialogSymbol = new javax.swing.JTextField();
ioiDialogSecurityID = new javax.swing.JTextField();
ioiDialogIDSource = new javax.swing.JComboBox();
ioiDialogNatural = new javax.swing.JComboBox();
ioiDialogShares = new javax.swing.JFormattedTextField();
ioiDialogPrice = new javax.swing.JFormattedTextField();
executionDialog = new javax.swing.JDialog();
executionDialogOK = new javax.swing.JButton();
executionDialogCancel = new javax.swing.JButton();
executionDialogShares = new javax.swing.JFormattedTextField();
executionDialogPrice = new javax.swing.JFormattedTextField();
executionSharesLabel = new javax.swing.JLabel();
executionPriceLabel = new javax.swing.JLabel();
messagePanel = new javax.swing.JPanel();
messageScrollPane = new javax.swing.JScrollPane();
messageTable = new javax.swing.JTable();
statusBarPanel = new javax.swing.JPanel();
executorRunningLabel = new javax.swing.JLabel();
ioiSenderRunningLabel = new javax.swing.JLabel();
clientConnectedLabel = new javax.swing.JLabel();
messageDetailPanel = new javax.swing.JPanel();
messageDetailScrollPane = new javax.swing.JScrollPane();
messageDetailTable = new javax.swing.JTable();
mainTabbedPane = new javax.swing.JTabbedPane();
loadPanel = new javax.swing.JPanel();
autoIOIPanel = new javax.swing.JPanel();
securityIDComboBox = new javax.swing.JComboBox();
rateSlider = new javax.swing.JSlider();
rateDisplayLable = new javax.swing.JLabel();
symbolLabel = new javax.swing.JLabel();
stopButton = new javax.swing.JButton();
startButton = new javax.swing.JButton();
symbolComboBox = new javax.swing.JComboBox();
securityIDLabel = new javax.swing.JLabel();
ioiSliderLabel = new javax.swing.JLabel();
autoExecutePanel = new javax.swing.JPanel();
stopExecutorButton = new javax.swing.JButton();
partialsSlider = new javax.swing.JSlider();
partialsLabel = new javax.swing.JLabel();
partialsNumber = new javax.swing.JLabel();
startExecutorButton = new javax.swing.JButton();
delayLabel = new javax.swing.JLabel();
executorDelay = new javax.swing.JComboBox();
ioiPanel = new javax.swing.JPanel();
manualIOIPanel = new javax.swing.JPanel();
```

```
singleIOIButton = new javax.swing.JButton();
cancelIOIButton = new javax.swing.JButton();
replaceIOIButton = new javax.swing.JButton();
ioiScrollPane = new javax.swing.JScrollPane();
ioiTable = new javax.swing.JTable();
orderPanel = new javax.swing.JPanel();
orderActionPanel = new javax.swing.JPanel();
acknowledgeButton = new javax.swing.JButton();
cancelButton = new javax.swing.JButton();
cancelPendingButton = new javax.swing.JButton();
replacePendingButton = new javax.swing.JButton();
executeButton = new javax.swing.JButton();
dfdButton = new javax.swing.JButton();
cancelAcceptButton = new javax.swing.JButton();
replaceAcceptButton = new javax.swing.JButton();
orderRejectButton = new javax.swing.JButton();
cancelRejectButton = new javax.swing.JButton();
replaceRejectButton = new javax.swing.JButton();
orderScrollPane = new javax.swing.JScrollPane();
orderTable = new javax.swing.JTable();
executionPanel = new javax.swing.JPanel();
executionActionPanel = new javax.swing.JPanel();
executionBustButton = new javax.swing.JButton();
executionCorrectButton = new javax.swing.JButton();
executionScrollPane = new javax.swing.JScrollPane();
executionTable = new javax.swing.JTable();
instrumentPanel = new javax.swing.JPanel();
instrumentScrollPane = new javax.swing.JScrollPane();
instrumentTable = new javax.swing.JTable();
reportPanel = new javax.swing.JPanel();
reportActionPanel = new javax.swing.JPanel();
customQueryRunButton = new javax.swing.JButton();
queryLabel = new javax.swing.JLabel();
queryText = new javax.swing.JTextField();
cannedQueryCombo = new javax.swing.JComboBox();
querySymbolLabel = new javax.swing.JLabel();
querySymbolText = new javax.swing.JTextField();
cannedQueryRunButton = new javax.swing.JButton();
reportScrollPane = new javax.swing.JScrollPane();
reportTable = new javax.swing.JTable();
settingsPanel = new javax.swing.JPanel();
autoResponsePanel = new javax.swing.JPanel();
autoAcknowledge = new javax.swing.JCheckBox();
autoPendingCancel = new javax.swing.JCheckBox();
autoPendingReplace = new javax.swing.JCheckBox();
autoCancel = new javax.swing.JCheckBox();
autoReplace = new javax.swing.JCheckBox();
cancelSeparator = new javax.swing.JSeparator();
replaceSeparator = new javax.swing.JSeparator();
saveSettingsButton = new javax.swing.JButton();
appSettingsPanel = new javax.swing.JPanel();
pricePrecisionLabel = new javax.swing.JLabel();
cachedObjectsLabel = new javax.swing.JLabel();
cachedObjectsCombo = new javax.swing.JComboBox();
pricePrecisionCombo = new javax.swing.JComboBox();
oboCompIDSeparator = new javax.swing.JSeparator();
sendOnBehalfOfCompID = new javax.swing.JCheckBox();
```

```
        sendOnBehalfOfSubID = new javax.swing.JCheckBox();
        oboCompIDSeparator1 = new javax.swing.JSeparator();
        logToFileLabel = new javax.swing.JLabel();
        logToFile = new javax.swing.JCheckBox();
        logToDB = new javax.swing.JCheckBox();
        showSettingsButton = new javax.swing.JButton();
        mainMenuBar = new javax.swing.JMenuBar();
        fileMenu = new javax.swing.JMenu();
        exitMenuItem = new javax.swing.JMenuItem();
        instrumentMenu = new javax.swing.JMenu();
        loadInstrumentMenuItem = new javax.swing.JMenuItem();
        helpMenu = new javax.swing.JMenu();
        aboutMenuItem = new javax.swing.JMenuItem();


aboutDialog.setDefaultCloseOperation(javax.swing.WindowConstants.DISPOS
E_ON_CLOSE);
        aboutDialog.setTitle("About...");
        aboutDialog.setLocationByPlatform(true);

        aboutPanel.setPreferredSize(new java.awt.Dimension(200, 100));

        okButton.setText("OK");
        okButton.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                okButtonActionPerformed(evt);
            }
        });

        aboutDialogLabel.setText("FIXimulator by Zoltan Feledy");

        javax.swing.GroupLayout aboutPanelLayout = new
javax.swing.GroupLayout(aboutPanel);
        aboutPanel.setLayout(aboutPanelLayout);
        aboutPanelLayout.setHorizontalGroup(

aboutPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(aboutPanelLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(aboutPanelLayout.createParallelGroup(javax.swing.GroupLayout.
Alignment.CENTER)
                    .addComponent(okButton)
                    .addComponent(aboutDialogLabel))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        aboutPanelLayout.setVerticalGroup(

aboutPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(aboutPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(aboutDialogLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 42,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(okButton)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        javax.swing.GroupLayout aboutDialogLayout = new
javax.swing.GroupLayout(aboutDialog.getContentPane());
        aboutDialog.getContentPane().setLayout(aboutDialogLayout);
        aboutDialogLayout.setHorizontalGroup(

aboutDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
            .addGroup(aboutDialogLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(aboutPanel,
javax.swing.GroupLayout.DEFAULT_SIZE, 157, Short.MAX_VALUE)
                .addContainerGap())
        );
        aboutDialogLayout.setVerticalGroup(

aboutDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
aboutDialogLayout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addComponent(aboutPanel,
javax.swing.GroupLayout.PREFERRED_SIZE, 93,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(7, 7, 7))
        );

        ioiDialog.setTitle("Add IOI...");
        ioiDialog.setAlwaysOnTop(true);
        ioiDialog.setName("ioiDialog"); // NOI18N

        ioiDialogOK.setText("OK");
        ioiDialogOK.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                ioiDialogOKActionPerformed(evt);
            }
        });

        ioiDialogCancel.setText("Cancel");
        ioiDialogCancel.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                ioiDialogCancelActionPerformed(evt);
            }
        });

        ioiIDLabel.setText("IOIid(23):");

        ioiSideLabel.setText("Side(54):");

        ioiSharesLabel.setText("IOIShares(27):");

        ioiSymbolLabel.setText("Symbol(55):");

        ioiSecurityIDLabel.setText("SecurityID(48):");

        ioiIDSourceLabel.setText("IDSource(22):");

        ioiPriceLabel.setText("Price(44):");

        ioiNaturalLabel.setText("IOINaturalFlag(130):");

        ioiDialogID.setText("ioiIDLabel");

        ioiDialogSide.setModel(new javax.swing.DefaultComboBoxModel(new
String[] { "Buy", "Sell", "Undisclosed" }));

        ioiDialogIDSource.setModel(new
javax.swing.DefaultComboBoxModel(new String[] { "CUSIP", "SEDOL",
"RIC", "TICKER", "OTHER" }));

        ioiDialogNatural.setModel(new
javax.swing.DefaultComboBoxModel(new String[] { "Yes", "No" }));

        ioiDialogShares.setFormatterFactory(new
javax.swing.text.DefaultFormatterFactory(new
javax.swing.text.NumberFormatter(new java.text.DecimalFormat("#0"))));

        ioiDialogPrice.setFormatterFactory(new
javax.swing.text.DefaultFormatterFactory(new
javax.swing.text.NumberFormatter(new
java.text.DecimalFormat("#,##0.####"))));
        ioiDialogPrice.setText("0.0");

        javax.swing.GroupLayout ioiDialogLayout = new
javax.swing.GroupLayout(ioiDialog.getContentPane());
        ioiDialog.getContentPane().setLayout(ioiDialogLayout);
        ioiDialogLayout.setHorizontalGroup(

ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(ioiDialogLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.TRAILING)
                    .addComponent(ioiDialogOK)
                    .addComponent(ioiNaturalLabel)
                    .addComponent(ioiIDLabel)
                    .addComponent(ioiSideLabel)
                    .addComponent(ioiSharesLabel)
                    .addComponent(ioiSymbolLabel)
                    .addComponent(ioiSecurityIDLabel)
                    .addComponent(ioiIDSourceLabel)
                    .addComponent(ioiPriceLabel))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addComponent(ioiDialogSide,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(ioiDialogSymbol,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(ioiDialogID)
                    .addComponent(ioiDialogSecurityID,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(ioiDialogIDSource,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(ioiDialogNatural,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(ioiDialogCancel)
                    .addComponent(ioiDialogShares,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(ioiDialogPrice,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(38, Short.MAX_VALUE))
        );

        ioiDialogLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
new java.awt.Component[] {ioiDialogCancel, ioiDialogOK});

        ioiDialogLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
new java.awt.Component[] {ioiIDLabel, ioiIDSourceLabel,
ioiNaturalLabel, ioiPriceLabel, ioiSecurityIDLabel, ioiSharesLabel,
ioiSideLabel, ioiSymbolLabel});

        ioiDialogLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
new java.awt.Component[] {ioiDialogID, ioiDialogIDSource,
ioiDialogNatural, ioiDialogPrice, ioiDialogSecurityID, ioiDialogShares,
ioiDialogSide, ioiDialogSymbol});

        ioiDialogLayout.setVerticalGroup(
ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(ioiDialogLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(ioiIDLabel)
                    .addComponent(ioiDialogID,
javax.swing.GroupLayout.PREFERRED_SIZE, 14,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(ioiSideLabel)
                    .addComponent(ioiDialogSide,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(ioiSharesLabel)
                    .addComponent(ioiDialogShares,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(ioiSymbolLabel)
                    .addComponent(ioiDialogSymbol,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(ioiSecurityIDLabel)
                    .addComponent(ioiDialogSecurityID,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
```

```
                    .addComponent(ioiIDSourceLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 14,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(ioiDialogIDSource,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(ioiPriceLabel)
                    .addComponent(ioiDialogPrice,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addComponent(ioiNaturalLabel)
                    .addComponent(ioiDialogNatural,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(32, 32, 32)

.addGroup(ioiDialogLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(ioiDialogOK)
                    .addComponent(ioiDialogCancel))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        executionDialog.setTitle("Add execution...");
        executionDialog.setName("executionDialog"); // NOI18N

        executionDialogOK.setText("OK");
        executionDialogOK.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                executionDialogOKActionPerformed(evt);
            }
        });

        executionDialogCancel.setText("Cancel");
        executionDialogCancel.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                executionDialogCancelActionPerformed(evt);
            }
        });
        executionDialogShares.setFormatterFactory(new
javax.swing.text.DefaultFormatterFactory(new
javax.swing.text.NumberFormatter(new java.text.DecimalFormat("#0"))));

        executionDialogPrice.setFormatterFactory(new
javax.swing.text.DefaultFormatterFactory(new
javax.swing.text.NumberFormatter(new
java.text.DecimalFormat("#,##0.####"))));
        executionDialogPrice.setText("0.0");

        executionSharesLabel.setText("LastShares(32):");

        executionPriceLabel.setText("LastPx(31):");

        javax.swing.GroupLayout executionDialogLayout = new
javax.swing.GroupLayout(executionDialog.getContentPane());

executionDialog.getContentPane().setLayout(executionDialogLayout);
        executionDialogLayout.setHorizontalGroup(

executionDialogLayout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
            .addGroup(executionDialogLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(executionDialogLayout.createParallelGroup(javax.swing.GroupLa
yout.Alignment.LEADING)

.addGroup(executionDialogLayout.createSequentialGroup()

.addGroup(executionDialogLayout.createParallelGroup(javax.swing.GroupLa
yout.Alignment.TRAILING)
                            .addComponent(executionPriceLabel)
                            .addComponent(executionSharesLabel))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(executionDialogLayout.createParallelGroup(javax.swing.GroupLa
yout.Alignment.LEADING)
                            .addComponent(executionDialogPrice,
javax.swing.GroupLayout.DEFAULT_SIZE, 92, Short.MAX_VALUE)
                            .addComponent(executionDialogShares,
javax.swing.GroupLayout.DEFAULT_SIZE, 92, Short.MAX_VALUE)))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
executionDialogLayout.createSequentialGroup()
                        .addComponent(executionDialogOK)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(executionDialogCancel)))
                .addContainerGap())
        );
        executionDialogLayout.setVerticalGroup(

executionDialogLayout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
```

```
            .addGroup(executionDialogLayout.createSequentialGroup()
                .addGap(19, 19, 19)

.addGroup(executionDialogLayout.createParallelGroup(javax.swing.GroupLa
yout.Alignment.BASELINE)
                    .addComponent(executionSharesLabel)
                    .addComponent(executionDialogShares,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(executionDialogLayout.createParallelGroup(javax.swing.GroupLa
yout.Alignment.BASELINE)
                    .addComponent(executionPriceLabel)
                    .addComponent(executionDialogPrice,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
18, Short.MAX_VALUE)

.addGroup(executionDialogLayout.createParallelGroup(javax.swing.GroupLa
yout.Alignment.BASELINE)
                    .addComponent(executionDialogOK)
                    .addComponent(executionDialogCancel))
                .addContainerGap())
        );


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("FIXimulator");
        setBounds(new java.awt.Rectangle(50, 50, 0, 0));
        setMinimumSize(new java.awt.Dimension(800, 600));
        setName("fiximulatorFrame"); // NOI18N
        setResizable(false);


messagePanel.setBorder(javax.swing.BorderFactory.createTitledBorder("Ap
plication Messages"));

        messageTable.setAutoCreateRowSorter(true);
        messageTable.setModel(new
edu.harvard.fas.zfeledy.fiximulator.ui.MessageTableModel());

messageTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        //Set initial column widths
        for (int i = 0; i < messageTable.getColumnCount(); i++){
            if (i==0)
            messageTable.getColumnModel().
            getColumn(i).setPreferredWidth(30);
            if (i==1)
            messageTable.getColumnModel().
            getColumn(i).setPreferredWidth(75);
            if (i==2)
            messageTable.getColumnModel().
            getColumn(i).setPreferredWidth(150);
            if (i==3)
            messageTable.getColumnModel().
            getColumn(i).setPreferredWidth(150);
            if (i==4)
            messageTable.getColumnModel().
            getColumn(i).setPreferredWidth(800);
        }
        messageScrollPane.setViewportView(messageTable);

        javax.swing.GroupLayout messagePanelLayout = new
javax.swing.GroupLayout(messagePanel);
        messagePanel.setLayout(messagePanelLayout);
        messagePanelLayout.setHorizontalGroup(

messagePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.LEADING)
            .addComponent(messageScrollPane,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 769, Short.MAX_VALUE)
        );
        messagePanelLayout.setVerticalGroup(

messagePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.LEADING)
            .addComponent(messageScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 166, Short.MAX_VALUE)
        );

        executorRunningLabel.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/edu/harvard/fas/zfeledy/
fiximulator/ui/red.gif"))); // NOI18N
        executorRunningLabel.setText("Executor status");

FIXimulator.getApplication().addStatusCallbacks(clientConnectedLabel,
ioiSenderRunningLabel, executorRunningLabel);

        ioiSenderRunningLabel.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/edu/harvard/fas/zfeledy/
fiximulator/ui/red.gif"))); // NOI18N
        ioiSenderRunningLabel.setText("IOI sender status");

        clientConnectedLabel.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/edu/harvard/fas/zfeledy/
fiximulator/ui/red.gif"))); // NOI18N
        if (FIXimulator.getApplication().getConnectionStatus())
        clientConnectedLabel.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/edu/harvard/fas/zfeledy/
fiximulator/ui/green.gif")));
        clientConnectedLabel.setText("Client connection status");

        javax.swing.GroupLayout statusBarPanelLayout = new
javax.swing.GroupLayout(statusBarPanel);
        statusBarPanel.setLayout(statusBarPanelLayout);
        statusBarPanelLayout.setHorizontalGroup(
statusBarPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
            .addGroup(statusBarPanelLayout.createSequentialGroup()
                .addGap(6, 6, 6)
                .addComponent(clientConnectedLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(ioiSenderRunningLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(executorRunningLabel)
                .addContainerGap(69, Short.MAX_VALUE))
        );
        statusBarPanelLayout.setVerticalGroup(

statusBarPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)

.addGroup(statusBarPanelLayout.createParallelGroup(javax.swing.GroupLay
out.Alignment.BASELINE)
                .addComponent(clientConnectedLabel)
                .addComponent(ioiSenderRunningLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 16,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(executorRunningLabel))
        );


messageDetailPanel.setBorder(javax.swing.BorderFactory.createTitledBord
er("Message Details"));

        messageDetailTable.setAutoCreateRowSorter(true);
        messageDetailTable.setModel(new
edu.harvard.fas.zfeledy.fiximulator.ui.MessageDetailTableModel(messageT
able));

messageDetailTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF
);
        for (int i = 0; i < messageDetailTable.getColumnCount(); i++){
            if (i==0)
            messageDetailTable.getColumnModel().
            getColumn(i).setPreferredWidth(100);
            if (i==1)
            messageDetailTable.getColumnModel().
            getColumn(i).setPreferredWidth(40);
            if (i==2)
            messageDetailTable.getColumnModel().
            getColumn(i).setPreferredWidth(150);
            if (i==3)
            messageDetailTable.getColumnModel().
            getColumn(i).setPreferredWidth(150);
        }
        messageDetailScrollPane.setViewportView(messageDetailTable);

        javax.swing.GroupLayout messageDetailPanelLayout = new
javax.swing.GroupLayout(messageDetailPanel);
```

```
        messageDetailPanel.setLayout(messageDetailPanelLayout);
        messageDetailPanelLayout.setHorizontalGroup(

messageDetailPanelLayout.createParallelGroup(javax.swing.GroupLayout.Al
ignment.LEADING)
            .addComponent(messageDetailScrollPane,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 317, Short.MAX_VALUE)
        );
        messageDetailPanelLayout.setVerticalGroup(

messageDetailPanelLayout.createParallelGroup(javax.swing.GroupLayout.Al
ignment.LEADING)
            .addGroup(messageDetailPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(messageDetailScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 298, Short.MAX_VALUE))
        );


autoIOIPanel.setBorder(javax.swing.BorderFactory.createTitledBorder("Au
tomated IOI Sender"));

        securityIDComboBox.setModel(new
javax.swing.DefaultComboBoxModel(new String[] { "RIC", "Sedol", "RIC",
"Cusip" }));
        securityIDComboBox.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                securityIDComboBoxActionPerformed(evt);
            }
        });

        rateSlider.setMajorTickSpacing(200);
        rateSlider.setMaximum(600);
        rateSlider.setMinorTickSpacing(50);
        rateSlider.setPaintLabels(true);
        rateSlider.setPaintTicks(true);
        rateSlider.setValue(60);
        rateSlider.addChangeListener(new
javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt)
{
                sliderChanged(evt);
            }
        });

        org.jdesktop.beansbinding.Binding binding =
org.jdesktop.beansbinding.Bindings.createAutoBinding(org.jdesktop.beans
binding.AutoBinding.UpdateStrategy.READ_WRITE, rateSlider,
org.jdesktop.beansbinding.ELProperty.create("${value}"),
rateDisplayLable,
org.jdesktop.beansbinding.BeanProperty.create("text"));
        bindingGroup.addBinding(binding);

        symbolLabel.setText("Symbol (55):");
        stopButton.setText("Stop");
        stopButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                stopButtonActionPerformed(evt);
            }
        });

        startButton.setText("Start");
        startButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                startButtonActionPerformed(evt);
            }
        });

        symbolComboBox.setModel(new
javax.swing.DefaultComboBoxModel(new String[] { "Ticker", "Sedol",
"RIC", "Cusip" }));
        symbolComboBox.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                symbolComboBoxActionPerformed(evt);
            }
        });

        securityIDLabel.setText("SecurityID (48):");

        ioiSliderLabel.setText(" IOIs per minute:");

        javax.swing.GroupLayout autoIOIPanelLayout = new
javax.swing.GroupLayout(autoIOIPanel);
        autoIOIPanel.setLayout(autoIOIPanelLayout);
        autoIOIPanelLayout.setHorizontalGroup(

autoIOIPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.LEADING)
            .addGroup(autoIOIPanelLayout.createSequentialGroup()

.addGroup(autoIOIPanelLayout.createParallelGroup(javax.swing.GroupLayou
t.Alignment.LEADING)

.addGroup(autoIOIPanelLayout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(startButton)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(stopButton))

.addGroup(autoIOIPanelLayout.createSequentialGroup()
                        .addComponent(ioiSliderLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 82,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(rateDisplayLable,
javax.swing.GroupLayout.PREFERRED_SIZE, 20,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGroup(autoIOIPanelLayout.createSequentialGroup()
                        .addComponent(symbolLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(symbolComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(18, 18, 18)
                        .addComponent(securityIDLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(securityIDComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addComponent(rateSlider,
javax.swing.GroupLayout.DEFAULT_SIZE, 409, Short.MAX_VALUE))
                .addContainerGap())
        );


autoIOIPanelLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL, new
java.awt.Component[] {startButton, stopButton});

        autoIOIPanelLayout.setVerticalGroup(

autoIOIPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
autoIOIPanelLayout.createSequentialGroup()

.addGroup(autoIOIPanelLayout.createParallelGroup(javax.swing.GroupLayou
t.Alignment.BASELINE)
                    .addComponent(symbolLabel)
                    .addComponent(symbolComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(securityIDLabel)
                    .addComponent(securityIDComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(autoIOIPanelLayout.createParallelGroup(javax.swing.GroupLayou
t.Alignment.BASELINE)
                    .addComponent(ioiSliderLabel)
                    .addComponent(rateDisplayLable,
javax.swing.GroupLayout.DEFAULT_SIZE, 14, Short.MAX_VALUE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(rateSlider,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(autoIOIPanelLayout.createParallelGroup(javax.swing.GroupLayou
t.Alignment.BASELINE)
                    .addComponent(startButton)
                    .addComponent(stopButton)))
        );


autoExecutePanel.setBorder(javax.swing.BorderFactory.createTitledBorder
("Automated Executor"));

        stopExecutorButton.setText("Stop");
        stopExecutorButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                stopExecutorButtonActionPerformed(evt);
            }
        });

        partialsSlider.setMajorTickSpacing(10);
        partialsSlider.setMaximum(50);
        partialsSlider.setPaintLabels(true);
        partialsSlider.setPaintTicks(true);
        partialsSlider.setValue(10);
        partialsSlider.addChangeListener(new
javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt)
{
                partialsSliderChanged(evt);
            }
        });

        partialsLabel.setText("Fills per order:");

        binding =
org.jdesktop.beansbinding.Bindings.createAutoBinding(org.jdesktop.beans
binding.AutoBinding.UpdateStrategy.READ_WRITE, partialsSlider,
org.jdesktop.beansbinding.ELProperty.create("${value}"),
partialsNumber, org.jdesktop.beansbinding.BeanProperty.create("text"));
        bindingGroup.addBinding(binding);

        startExecutorButton.setText("Start");
        startExecutorButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
```

```
                startExecutorButtonActionPerformed(evt);
            }
        });

        delayLabel.setText("Delay:");

        executorDelay.setModel(new javax.swing.DefaultComboBoxModel(new
String[] { "1 ms", "10 ms", "100 ms", "1 second", "5 seconds" }));
        executorDelay.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                executorDelayActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout autoExecutePanelLayout = new
javax.swing.GroupLayout(autoExecutePanel);
        autoExecutePanel.setLayout(autoExecutePanelLayout);
        autoExecutePanelLayout.setHorizontalGroup(

autoExecutePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alig
nment.LEADING)
            .addGroup(autoExecutePanelLayout.createSequentialGroup()

.addGroup(autoExecutePanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.LEADING)

.addGroup(autoExecutePanelLayout.createSequentialGroup()
                        .addContainerGap()

.addGroup(autoExecutePanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.LEADING)

.addGroup(autoExecutePanelLayout.createSequentialGroup()
                                .addComponent(delayLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(executorDelay,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addGap(18, 18, 18)
                                .addComponent(partialsLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(partialsNumber))

.addGroup(autoExecutePanelLayout.createSequentialGroup()
                                .addComponent(startExecutorButton)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(stopExecutorButton))))
                    .addComponent(partialsSlider,
javax.swing.GroupLayout.DEFAULT_SIZE, 409, Short.MAX_VALUE))
                .addContainerGap())
        );
```

```
autoExecutePanelLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
new java.awt.Component[] {startExecutorButton, stopExecutorButton});

        autoExecutePanelLayout.setVerticalGroup(

autoExecutePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alig
nment.LEADING)
            .addGroup(autoExecutePanelLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(autoExecutePanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.BASELINE)
                    .addComponent(delayLabel)
                    .addComponent(executorDelay,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(partialsNumber)
                    .addComponent(partialsLabel))
                .addGap(11, 11, 11)
                .addComponent(partialsSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, 44,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(autoExecutePanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.BASELINE)
                    .addComponent(stopExecutorButton)
                    .addComponent(startExecutorButton))
                .addGap(14, 14, 14))
        );

        javax.swing.GroupLayout loadPanelLayout = new
javax.swing.GroupLayout(loadPanel);
        loadPanel.setLayout(loadPanelLayout);
        loadPanelLayout.setHorizontalGroup(

loadPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
loadPanelLayout.createSequentialGroup()

.addGroup(loadPanelLayout.createParallelGroup(javax.swing.GroupLayout.A
lignment.TRAILING)
                    .addComponent(autoExecutePanel,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(autoIOIPanel,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap())
        );
        loadPanelLayout.setVerticalGroup(
loadPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(loadPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(autoIOIPanel,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(autoExecutePanel,
javax.swing.GroupLayout.PREFERRED_SIZE, 141,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        mainTabbedPane.addTab("Load", loadPanel);


manualIOIPanel.setBorder(javax.swing.BorderFactory.createTitledBorder("
IOIs"));

        singleIOIButton.setText("Add IOI");
        singleIOIButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                singleIOIButtonActionPerformed(evt);
            }
        });

        cancelIOIButton.setText("Cancel IOI");
        cancelIOIButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                cancelIOIButtonActionPerformed(evt);
            }
        });

        replaceIOIButton.setText("Replace IOI");
        replaceIOIButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                replaceIOIButtonActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout manualIOIPanelLayout = new
javax.swing.GroupLayout(manualIOIPanel);
        manualIOIPanel.setLayout(manualIOIPanelLayout);
        manualIOIPanelLayout.setHorizontalGroup(
manualIOIPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
            .addGroup(manualIOIPanelLayout.createSequentialGroup()
                .addComponent(singleIOIButton)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(replaceIOIButton)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(cancelIOIButton)
                .addContainerGap(140, Short.MAX_VALUE))
        );


manualIOIPanelLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
new java.awt.Component[] {cancelIOIButton, replaceIOIButton,
singleIOIButton});

        manualIOIPanelLayout.setVerticalGroup(

manualIOIPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)

.addGroup(manualIOIPanelLayout.createParallelGroup(javax.swing.GroupLay
out.Alignment.BASELINE)
                .addComponent(singleIOIButton)
                .addComponent(replaceIOIButton)
                .addComponent(cancelIOIButton))
        );

        ioiTable.setDefaultRenderer(Object.class, new
IOICellRenderer());
        ioiTable.setAutoCreateRowSorter(true);
        ioiTable.setModel(new
edu.harvard.fas.zfeledy.fiximulator.ui.IOITableModel());
        ioiTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        //Set initial column widths
        for (int i = 0; i < ioiTable.getColumnCount(); i++){
            if (i==0)
            ioiTable.getColumnModel().
            getColumn(i).setPreferredWidth(100);
            if (i==1)
            ioiTable.getColumnModel().
            getColumn(i).setPreferredWidth(60);
            if (i==2)
            ioiTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==3)
            ioiTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==4)
            ioiTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==5)
            ioiTable.getColumnModel().
            getColumn(i).setPreferredWidth(60);
            if (i==6)
            ioiTable.getColumnModel().
            getColumn(i).setPreferredWidth(60);
            if (i==7)
            ioiTable.getColumnModel().
            getColumn(i).setPreferredWidth(60);
            if (i==8)
            ioiTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==9)
            ioiTable.getColumnModel().
            getColumn(i).setPreferredWidth(100);
        }

ioiTable.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTI
ON);
        ioiScrollPane.setViewportView(ioiTable);

        javax.swing.GroupLayout ioiPanelLayout = new
javax.swing.GroupLayout(ioiPanel);
        ioiPanel.setLayout(ioiPanelLayout);
        ioiPanelLayout.setHorizontalGroup(

ioiPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
ioiPanelLayout.createSequentialGroup()
                .addComponent(manualIOIPanel,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())

.addGroup(ioiPanelLayout.createParallelGroup(javax.swing.GroupLayout.Al
ignment.LEADING)
                .addComponent(ioiScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 441, Short.MAX_VALUE))
        );
        ioiPanelLayout.setVerticalGroup(

ioiPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
            .addGroup(ioiPanelLayout.createSequentialGroup()
                .addComponent(manualIOIPanel,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(258, Short.MAX_VALUE))

.addGroup(ioiPanelLayout.createParallelGroup(javax.swing.GroupLayout.Al
ignment.LEADING)
                .addGroup(ioiPanelLayout.createSequentialGroup()
                    .addGap(54, 54, 54)
                    .addComponent(ioiScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 254, Short.MAX_VALUE)))
        );

        mainTabbedPane.addTab("IOIs", ioiPanel);
```

```
orderActionPanel.setBorder(javax.swing.BorderFactory.createTitledBorder
("Orders"));

        acknowledgeButton.setText("Acknowledge");
        acknowledgeButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                acknowledgeButtonActionPerformed(evt);
            }
        });

        cancelButton.setText("Cancel");
        cancelButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                cancelButtonActionPerformed(evt);
            }
        });

        cancelPendingButton.setText("Pending Cancel");
        cancelPendingButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                cancelPendingButtonActionPerformed(evt);
            }
        });

        replacePendingButton.setText("Pending Replace");
        replacePendingButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                replacePendingButtonActionPerformed(evt);
            }
        });

        executeButton.setText("Execute");
        executeButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                executeButtonActionPerformed(evt);
            }
        });

        dfdButton.setText("DFD");
        dfdButton.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                dfdButtonActionPerformed(evt);
            }
        });

        cancelAcceptButton.setText("Accept Cancel");
        cancelAcceptButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                cancelAcceptButtonActionPerformed(evt);
            }
        });

        replaceAcceptButton.setText("Accept Replace");
        replaceAcceptButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                replaceAcceptButtonActionPerformed(evt);
            }
        });

        orderRejectButton.setText("Reject");
        orderRejectButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                orderRejectButtonActionPerformed(evt);
            }
        });

        cancelRejectButton.setText("Reject Cancel");
        cancelRejectButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                cancelRejectButtonActionPerformed(evt);
            }
        });

        replaceRejectButton.setText("Reject Replace");
        replaceRejectButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                replaceRejectButtonActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout orderActionPanelLayout = new
javax.swing.GroupLayout(orderActionPanel);
        orderActionPanel.setLayout(orderActionPanelLayout);
        orderActionPanelLayout.setHorizontalGroup(

orderActionPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alig
nment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
orderActionPanelLayout.createSequentialGroup()
                .addContainerGap()
```

```
.addGroup(orderActionPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.CENTER)

.addGroup(orderActionPanelLayout.createSequentialGroup()
                        .addComponent(orderRejectButton,
javax.swing.GroupLayout.DEFAULT_SIZE, 101, Short.MAX_VALUE)
                        .addGap(6, 6, 6)
                        .addComponent(dfdButton,
javax.swing.GroupLayout.PREFERRED_SIZE, 65,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGroup(orderActionPanelLayout.createSequentialGroup()
                        .addComponent(acknowledgeButton,
javax.swing.GroupLayout.PREFERRED_SIZE, 95,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(6, 6, 6)
                        .addComponent(cancelButton))
                    .addComponent(executeButton,
javax.swing.GroupLayout.PREFERRED_SIZE, 171,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(orderActionPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.CENTER)
                    .addComponent(cancelRejectButton)
                    .addComponent(cancelAcceptButton)
                    .addComponent(cancelPendingButton))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(orderActionPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.CENTER)
                    .addComponent(replacePendingButton)
                    .addComponent(replaceAcceptButton)
                    .addComponent(replaceRejectButton))
                .addContainerGap())
        );


orderActionPanelLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
new java.awt.Component[] {acknowledgeButton, orderRejectButton});


orderActionPanelLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
new java.awt.Component[] {replaceAcceptButton, replacePendingButton,
replaceRejectButton});


orderActionPanelLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
new java.awt.Component[] {cancelAcceptButton, cancelPendingButton,
cancelRejectButton});


orderActionPanelLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
new java.awt.Component[] {cancelButton, dfdButton});
```

```
        orderActionPanelLayout.setVerticalGroup(

orderActionPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alig
nment.LEADING)
            .addGroup(orderActionPanelLayout.createSequentialGroup()

.addGroup(orderActionPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.LEADING)

.addGroup(orderActionPanelLayout.createSequentialGroup()

.addGroup(orderActionPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.LEADING)
                            .addComponent(cancelPendingButton)

.addGroup(orderActionPanelLayout.createSequentialGroup()
                                .addGap(29, 29, 29)
                                .addComponent(cancelAcceptButton))

.addGroup(orderActionPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.BASELINE)
                                .addComponent(cancelButton)
                                .addComponent(acknowledgeButton))

.addGroup(orderActionPanelLayout.createSequentialGroup()
                                .addGap(29, 29, 29)

.addGroup(orderActionPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.BASELINE)
                                    .addComponent(orderRejectButton)
                                    .addComponent(dfdButton))))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(orderActionPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.BASELINE)
                            .addComponent(cancelRejectButton)
                            .addComponent(replaceRejectButton)
                            .addComponent(executeButton)))

.addGroup(orderActionPanelLayout.createSequentialGroup()
                        .addComponent(replacePendingButton)
                        .addGap(6, 6, 6)
                        .addComponent(replaceAcceptButton)))
                .addContainerGap())
        );

        //ioiTable.setDefaultRenderer(Object.class, new
IOICellRenderer());
        orderTable.setAutoCreateRowSorter(true);
        orderTable.setModel(new
edu.harvard.fas.zfeledy.fiximulator.ui.OrderTableModel());

orderTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        //Set initial column widths
        for (int i = 0; i < orderTable.getColumnCount(); i++){
            if (i==0) // Order ID
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(90);
            if (i==1) // Status
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(100);
            if (i==2) // Side
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(30);
            if (i==3)
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(60);
            if (i==4)
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==5)
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(80);
            if (i==6) // Limit
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==7) // TIF
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(30);
            if (i==8) // Executed
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(70);
            if (i==9) // Open
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==10) // AvgPx
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==11) // ClOrdID
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(90);
            if (i==12) // OrigClOrdID
            orderTable.getColumnModel().
            getColumn(i).setPreferredWidth(90);
        }

orderTable.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELEC
TION);
        orderScrollPane.setViewportView(orderTable);

        javax.swing.GroupLayout orderPanelLayout = new
javax.swing.GroupLayout(orderPanel);
        orderPanel.setLayout(orderPanelLayout);
        orderPanelLayout.setHorizontalGroup(

orderPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addComponent(orderScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 442, Short.MAX_VALUE)
            .addGroup(orderPanelLayout.createSequentialGroup()
                .addComponent(orderActionPanel,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())
        );
        orderPanelLayout.setVerticalGroup(

orderPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(orderPanelLayout.createSequentialGroup()
                .addComponent(orderActionPanel,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(orderScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 183, Short.MAX_VALUE))
        );

        mainTabbedPane.addTab("Orders", orderPanel);


executionActionPanel.setBorder(javax.swing.BorderFactory.createTitledBo
rder("Executions"));

        executionBustButton.setText("Bust");
        executionBustButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                executionBustButtonActionPerformed(evt);
            }
        });

        executionCorrectButton.setText("Correct");
        executionCorrectButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                executionCorrectButtonActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout executionActionPanelLayout = new
javax.swing.GroupLayout(executionActionPanel);
        executionActionPanel.setLayout(executionActionPanelLayout);
        executionActionPanelLayout.setHorizontalGroup(

executionActionPanelLayout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)

.addGroup(executionActionPanelLayout.createSequentialGroup()
                .addComponent(executionBustButton)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(executionCorrectButton)
                .addGap(228, 228, 228))
        );
        executionActionPanelLayout.setVerticalGroup(

executionActionPanelLayout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)

.addGroup(executionActionPanelLayout.createParallelGroup(javax.swing.Gr
oupLayout.Alignment.BASELINE)
                .addComponent(executionBustButton)
                .addComponent(executionCorrectButton))
        );

        executionTable.setDefaultRenderer(Object.class, new
ExecutionCellRenderer());
        executionTable.setAutoCreateRowSorter(true);
        executionTable.setModel(new
edu.harvard.fas.zfeledy.fiximulator.ui.ExecutionTableModel());

executionTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        //Set initial column widths
        for (int i = 0; i < executionTable.getColumnCount(); i++){
            if (i==0) // ID
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(90);
            if (i==1) // ClOrdID
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(90);
            if (i==2) // Side
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(30);
            if (i==3) // Sybol
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(60);
            if (i==4) // LastQty
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==5) // LastPx
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==6) // CumQty
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==7) // AvgPx
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==8) // Open
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==9) // ExecType
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(70);
            if (i==10) // ExecTransType
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(70);
            if (i==11) // RefID
            executionTable.getColumnModel().
            getColumn(i).setPreferredWidth(90);
        }

executionTable.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_S
ELECTION);
        executionScrollPane.setViewportView(executionTable);

        javax.swing.GroupLayout executionPanelLayout = new
javax.swing.GroupLayout(executionPanel);
        executionPanel.setLayout(executionPanelLayout);
        executionPanelLayout.setHorizontalGroup(

executionPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
            .addComponent(executionActionPanel,
javax.swing.GroupLayout.DEFAULT_SIZE, 441, Short.MAX_VALUE)
            .addComponent(executionScrollPane, 0, 0, Short.MAX_VALUE)
        );
        executionPanelLayout.setVerticalGroup(

executionPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
            .addGroup(executionPanelLayout.createSequentialGroup()
                .addComponent(executionActionPanel,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(executionScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 252, Short.MAX_VALUE))
        );

        mainTabbedPane.addTab("Executions", executionPanel);

        instrumentTable.setAutoCreateRowSorter(true);
        instrumentTable.setModel(new
edu.harvard.fas.zfeledy.fiximulator.ui.InstrumentTableModel());

instrumentTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        //Set initial column widths
        for (int i = 0; i < instrumentTable.getColumnCount(); i++){
            if (i==0)
            instrumentTable.getColumnModel().
            getColumn(i).setPreferredWidth(50);
            if (i==1)
            instrumentTable.getColumnModel().
            getColumn(i).setPreferredWidth(200);
        }

instrumentTable.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_
SELECTION);
        instrumentScrollPane.setViewportView(instrumentTable);

        javax.swing.GroupLayout instrumentPanelLayout = new
javax.swing.GroupLayout(instrumentPanel);
        instrumentPanel.setLayout(instrumentPanelLayout);
        instrumentPanelLayout.setHorizontalGroup(

instrumentPanelLayout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
            .addComponent(instrumentScrollPane,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 441, Short.MAX_VALUE)
        );
        instrumentPanelLayout.setVerticalGroup(

instrumentPanelLayout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
            .addGroup(instrumentPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(instrumentScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 297, Short.MAX_VALUE))
        );

        mainTabbedPane.addTab("Instruments", instrumentPanel);


reportActionPanel.setBorder(javax.swing.BorderFactory.createTitledBorde
r("Reporting"));

        customQueryRunButton.setText("Run");
        customQueryRunButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                customQueryRunButtonActionPerformed(evt);
            }
        });

        queryLabel.setText("Query:");

        queryText.setText("select text from messages_log where text
like '%35=6%';");

        cannedQueryCombo.setModel(new
javax.swing.DefaultComboBoxModel(new String[] { "Show last 10 IOIs...",
"Show last 10 orders...", "Show last 10 executions...", "Show all IOIs
where Symbol(55) is...", "Show all orders where Symbol(55) is...",
"Show all executions where Symbol(55) is...", "Show all activity where
Symbol(55) is..." }));

        querySymbolLabel.setText("Symbol:");

        cannedQueryRunButton.setText("Run");
        cannedQueryRunButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                cannedQueryRunButtonActionPerformed(evt);
            }
        });
        javax.swing.GroupLayout reportActionPanelLayout = new
javax.swing.GroupLayout(reportActionPanel);
        reportActionPanel.setLayout(reportActionPanelLayout);
        reportActionPanelLayout.setHorizontalGroup(

reportActionPanelLayout.createParallelGroup(javax.swing.GroupLayout.Ali
gnment.LEADING)
            .addGroup(reportActionPanelLayout.createSequentialGroup()

.addGroup(reportActionPanelLayout.createParallelGroup(javax.swing.Group
Layout.Alignment.LEADING)

.addGroup(reportActionPanelLayout.createSequentialGroup()
                        .addComponent(cannedQueryCombo,
javax.swing.GroupLayout.PREFERRED_SIZE, 276,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(querySymbolLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(querySymbolText,
javax.swing.GroupLayout.DEFAULT_SIZE, 50, Short.MAX_VALUE))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
reportActionPanelLayout.createSequentialGroup()
                        .addComponent(queryLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(queryText,
javax.swing.GroupLayout.PREFERRED_SIZE, 334,
javax.swing.GroupLayout.PREFERRED_SIZE)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(reportActionPanelLayout.createParallelGroup(javax.swing.Group
Layout.Alignment.LEADING)
                    .addComponent(cannedQueryRunButton)
                    .addComponent(customQueryRunButton))
                .addContainerGap())
        );
        reportActionPanelLayout.setVerticalGroup(

reportActionPanelLayout.createParallelGroup(javax.swing.GroupLayout.Ali
gnment.LEADING)
            .addGroup(reportActionPanelLayout.createSequentialGroup()

.addGroup(reportActionPanelLayout.createParallelGroup(javax.swing.Group
Layout.Alignment.BASELINE)
                    .addComponent(queryLabel)
                    .addComponent(customQueryRunButton)
                    .addComponent(queryText,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(reportActionPanelLayout.createParallelGroup(javax.swing.Group
Layout.Alignment.BASELINE)
                    .addComponent(cannedQueryRunButton)
                    .addComponent(querySymbolText,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(querySymbolLabel)
                    .addComponent(cannedQueryCombo,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap())
        );

        executionTable.setDefaultRenderer(Object.class, new
ExecutionCellRenderer());
        reportTable.setAutoCreateRowSorter(true);
        reportTable.setModel(new QueryTableModel());

reportTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);

reportTable.setSelectionMode(javax.swing.ListSelectionModel.MULTIPLE_IN
TERVAL_SELECTION);
        reportScrollPane.setViewportView(reportTable);

        javax.swing.GroupLayout reportPanelLayout = new
javax.swing.GroupLayout(reportPanel);
        reportPanel.setLayout(reportPanelLayout);
        reportPanelLayout.setHorizontalGroup(

reportPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
reportPanelLayout.createSequentialGroup()

.addGroup(reportPanelLayout.createParallelGroup(javax.swing.GroupLayout
.Alignment.TRAILING)
                    .addComponent(reportScrollPane,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 441, Short.MAX_VALUE)
                    .addComponent(reportActionPanel,
javax.swing.GroupLayout.DEFAULT_SIZE, 441, Short.MAX_VALUE))
                .addContainerGap())
        );
        reportPanelLayout.setVerticalGroup(

reportPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
reportPanelLayout.createSequentialGroup()
                .addComponent(reportActionPanel,
javax.swing.GroupLayout.DEFAULT_SIZE, 78, Short.MAX_VALUE)
```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(reportScrollPane,
javax.swing.GroupLayout.PREFERRED_SIZE, 213,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        mainTabbedPane.addTab("Reports", reportPanel);


autoResponsePanel.setBorder(javax.swing.BorderFactory.createTitledBorde
r("Automated Responses"));

        autoAcknowledge.setText("Acknowledge orders on receipt");
        try {
            autoAcknowledge.setSelected(
                FIXimulator.getApplication().getSettings()
                .getBool("FIXimulatorAutoAcknowledge"));
        } catch ( Exception e ){}
        autoAcknowledge.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                autoAcknowledgeActionPerformed(evt);
            }
        });

        autoPendingCancel.setText("Send Pending Cancel");
        try {
            autoPendingCancel.setSelected(
                FIXimulator.getApplication().getSettings()
                .getBool("FIXimulatorAutoPendingCancel"));
        } catch ( Exception e ){}
        autoPendingCancel.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                autoPendingCancelActionPerformed(evt);
            }
        });

        autoPendingReplace.setText("Send Pending Replace");
        try {
            autoPendingReplace.setSelected(
                FIXimulator.getApplication().getSettings()
                .getBool("FIXimulatorAutoPendingReplace"));
        } catch ( Exception e ){}
        autoPendingReplace.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                autoPendingReplaceActionPerformed(evt);
            }
        });

        autoCancel.setText("Accept order cancellations");
        try {
            autoCancel.setSelected(
                FIXimulator.getApplication().getSettings()
                .getBool("FIXimulatorAutoCancel"));
        } catch ( Exception e ){}
        autoCancel.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                autoCancelActionPerformed(evt);
            }
        });

        autoReplace.setText("Accept order replacements");
        try {
            autoReplace.setSelected(
                FIXimulator.getApplication().getSettings()
                .getBool("FIXimulatorAutoReplace"));
        } catch ( Exception e ){}
        autoReplace.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                autoReplaceActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout autoResponsePanelLayout = new
javax.swing.GroupLayout(autoResponsePanel);
        autoResponsePanel.setLayout(autoResponsePanelLayout);
        autoResponsePanelLayout.setHorizontalGroup(

autoResponsePanelLayout.createParallelGroup(javax.swing.GroupLayout.Ali
gnment.LEADING)
            .addGroup(autoResponsePanelLayout.createSequentialGroup()

.addGroup(autoResponsePanelLayout.createParallelGroup(javax.swing.Group
Layout.Alignment.LEADING)
                    .addComponent(autoPendingCancel)
                    .addComponent(autoCancel)
                    .addComponent(cancelSeparator,
javax.swing.GroupLayout.PREFERRED_SIZE, 177,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(autoAcknowledge)
                    .addComponent(replaceSeparator,
javax.swing.GroupLayout.PREFERRED_SIZE, 177,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(autoPendingReplace)
                    .addComponent(autoReplace))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        autoResponsePanelLayout.setVerticalGroup(

autoResponsePanelLayout.createParallelGroup(javax.swing.GroupLayout.Ali
gnment.LEADING)
            .addGroup(autoResponsePanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(autoAcknowledge)
                .addGap(6, 6, 6)
                .addComponent(cancelSeparator,
javax.swing.GroupLayout.PREFERRED_SIZE, 10,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(autoPendingCancel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(autoCancel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(replaceSeparator,
javax.swing.GroupLayout.PREFERRED_SIZE, 10,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(autoPendingReplace)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(autoReplace)
                .addContainerGap(76, Short.MAX_VALUE))
        );

        saveSettingsButton.setText("Save Settings");
        saveSettingsButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                saveSettingsButtonActionPerformed(evt);
            }
        });


appSettingsPanel.setBorder(javax.swing.BorderFactory.createTitledBorder
("FIXimulator Settings"));

        pricePrecisionLabel.setText("Price precision:");

        cachedObjectsLabel.setText("Number of cached objects:");

        cachedObjectsCombo.setModel(new
javax.swing.DefaultComboBoxModel(new String[] { "50", "100", "200" }));
        try {
            Long settingValue =
FIXimulator.getApplication().getSettings()
            .getLong("FIXimulatorCachedObjects");
            if ( settingValue == 50 || settingValue == 100 ||
settingValue == 200) {

cachedObjectsCombo.setSelectedItem(settingValue.toString());
            } else {
                // default due to bad value
                cachedObjectsCombo.setSelectedItem("50");
                FIXimulator.getApplication().getSettings()
                .setLong("FIXimulatorCachedObjects", 50);
            }
        } catch ( Exception e ){
            // default to to setting not existing
            cachedObjectsCombo.setSelectedItem("50");
            FIXimulator.getApplication().getSettings()
            .setLong("FIXimulatorCachedObjects", 50);
        }
        cachedObjectsCombo.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                cachedObjectsComboActionPerformed(evt);
            }
        });

        pricePrecisionCombo.setModel(new
javax.swing.DefaultComboBoxModel(new String[] { "0", "1", "2", "3",
"4", "5", "6", "7", "8", "9" }));
        try {
            Long settingValue =
FIXimulator.getApplication().getSettings()
            .getLong("FIXimulatorPricePrecision");
            if ( settingValue >= 0 && settingValue < 10 ) {

pricePrecisionCombo.setSelectedItem(settingValue.toString());
            } else {
                // default due to bad value
                pricePrecisionCombo.setSelectedItem("4");
                FIXimulator.getApplication().getSettings()
                .setLong("FIXimulatorPricePrecision", 4);
            }
        } catch ( Exception e ){
            // default to to setting not existing
            pricePrecisionCombo.setSelectedItem("4");
            FIXimulator.getApplication().getSettings()
            .setLong("FIXimulatorPricePrecision", 4);
        }
        pricePrecisionCombo.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                pricePrecisionComboActionPerformed(evt);
            }
        });

        sendOnBehalfOfCompID.setText("Send OnBehalfOfCompID (115)");
        try {
            sendOnBehalfOfCompID.setSelected(
                FIXimulator.getApplication().getSettings()
                .getBool("FIXimulatorSendOnBehalfOfCompID"));
        } catch ( Exception e ){}
        sendOnBehalfOfCompID.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                sendOnBehalfOfCompIDActionPerformed(evt);
            }
        });

        sendOnBehalfOfSubID.setText("Send OnBehalfOfSubID (116)");
        try {
            sendOnBehalfOfSubID.setSelected(
                FIXimulator.getApplication().getSettings()
                .getBool("FIXimulatorSendOnBehalfOfSubID"));
        } catch ( Exception e ){}
        sendOnBehalfOfSubID.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                sendOnBehalfOfSubIDActionPerformed(evt);
            }
        });

        logToFileLabel.setText("<html>Changing the logging requires
saving the settings and restarting the application...</htm;>");

        logToFile.setText("Log to file");
        try {
            logToFile.setSelected(
                FIXimulator.getApplication().getSettings()
                .getBool("FIXimulatorLogToFile"));
        } catch ( Exception e ){}
        logToFile.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                logToFileActionPerformed(evt);
            }
        });

        logToDB.setText("Log to database");
        try {
            logToDB.setSelected(
                FIXimulator.getApplication().getSettings()
                .getBool("FIXimulatorLogToDB"));
        } catch ( Exception e ){}
        logToDB.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                logToDBActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout appSettingsPanelLayout = new
javax.swing.GroupLayout(appSettingsPanel);
        appSettingsPanel.setLayout(appSettingsPanelLayout);
        appSettingsPanelLayout.setHorizontalGroup(

appSettingsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alig
nment.LEADING)
            .addGroup(appSettingsPanelLayout.createSequentialGroup()
                .addContainerGap()
```

```
.addGroup(appSettingsPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.LEADING)
                    .addComponent(logToFileLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 197,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup(appSettingsPanelLayout.createSequentialGroup()
                        .addComponent(logToFile)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(logToDB))
                    .addComponent(oboCompIDSeparator1,
javax.swing.GroupLayout.DEFAULT_SIZE, 197, Short.MAX_VALUE)
                    .addComponent(sendOnBehalfOfSubID)

.addGroup(appSettingsPanelLayout.createSequentialGroup()

.addGroup(appSettingsPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.LEADING)
                            .addComponent(cachedObjectsLabel)
                            .addComponent(pricePrecisionLabel))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(appSettingsPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.LEADING)
                            .addComponent(cachedObjectsCombo,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(pricePrecisionCombo,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                    .addComponent(oboCompIDSeparator,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 197, Short.MAX_VALUE)
                    .addComponent(sendOnBehalfOfCompID))
                .addContainerGap())
        );


appSettingsPanelLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
new java.awt.Component[] {cachedObjectsCombo, pricePrecisionCombo});

        appSettingsPanelLayout.setVerticalGroup(

appSettingsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alig
nment.LEADING)
            .addGroup(appSettingsPanelLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(appSettingsPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.BASELINE)
                    .addComponent(pricePrecisionLabel)
                    .addComponent(pricePrecisionCombo,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(appSettingsPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.BASELINE)
                    .addComponent(cachedObjectsLabel)
                    .addComponent(cachedObjectsCombo,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)
                .addComponent(oboCompIDSeparator,
javax.swing.GroupLayout.PREFERRED_SIZE, 10,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(sendOnBehalfOfCompID)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(sendOnBehalfOfSubID)
                .addGap(12, 12, 12)
                .addComponent(oboCompIDSeparator1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(logToFileLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, 45, Short.MAX_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(appSettingsPanelLayout.createParallelGroup(javax.swing.GroupL
ayout.Alignment.BASELINE)
                    .addComponent(logToFile)
                    .addComponent(logToDB))
                .addContainerGap())
        );

        showSettingsButton.setText("Show Settings");
        showSettingsButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                showSettingsButtonActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout settingsPanelLayout = new
javax.swing.GroupLayout(settingsPanel);
        settingsPanel.setLayout(settingsPanelLayout);
        settingsPanelLayout.setHorizontalGroup(
settingsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
            .addGroup(settingsPanelLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(settingsPanelLayout.createParallelGroup(javax.swing.GroupLayo
ut.Alignment.LEADING)

.addGroup(settingsPanelLayout.createSequentialGroup()
                        .addComponent(autoResponsePanel,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(appSettingsPanel,
javax.swing.GroupLayout.DEFAULT_SIZE, 219, Short.MAX_VALUE))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
settingsPanelLayout.createSequentialGroup()
                        .addComponent(showSettingsButton)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(saveSettingsButton)))
                .addContainerGap())
        );


settingsPanelLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL, new
java.awt.Component[] {saveSettingsButton, showSettingsButton});

        settingsPanelLayout.setVerticalGroup(

settingsPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
settingsPanelLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(settingsPanelLayout.createParallelGroup(javax.swing.GroupLayo
ut.Alignment.LEADING)
                    .addComponent(appSettingsPanel,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(autoResponsePanel,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(settingsPanelLayout.createParallelGroup(javax.swing.GroupLayo
ut.Alignment.BASELINE)
                    .addComponent(saveSettingsButton)
                    .addComponent(showSettingsButton))
                .addContainerGap())
        );
        mainTabbedPane.addTab("Settings", settingsPanel);

        fileMenu.setText("File");

        exitMenuItem.setText("Exit");
        exitMenuItem.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                exitMenuItemActionPerformed(evt);
            }
        });
        fileMenu.add(exitMenuItem);

        mainMenuBar.add(fileMenu);

        instrumentMenu.setLabel("Instruments");

        loadInstrumentMenuItem.setText("Load Instruments...");
        loadInstrumentMenuItem.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                loadInstrumentMenuItemActionPerformed(evt);
            }
        });
        instrumentMenu.add(loadInstrumentMenuItem);

        mainMenuBar.add(instrumentMenu);

        helpMenu.setLabel("Help");

        aboutMenuItem.setText("About...");
        aboutMenuItem.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                aboutMenuItemActionPerformed(evt);
            }
        });
        helpMenu.add(aboutMenuItem);

        mainMenuBar.add(helpMenu);

        setJMenuBar(mainMenuBar);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
                    .addComponent(messagePanel,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                        .addComponent(mainTabbedPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 446, Short.MAX_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(messageDetailPanel,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addComponent(statusBarPanel,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap())
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
TRAILING)
                    .addComponent(messageDetailPanel,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(mainTabbedPane,
javax.swing.GroupLayout.PREFERRED_SIZE, 336,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(messagePanel,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(statusBarPanel,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        bindingGroup.bind();

        pack();
    }// </editor-fold>
```

```
private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
    aboutDialog.dispose();
}

private void symbolComboBoxActionPerformed(java.awt.event.ActionEvent
evt) {
    FIXimulator.getApplication().setNewSymbol(
            symbolComboBox.getSelectedItem().toString());
}

private void
securityIDComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
    FIXimulator.getApplication().setNewSecurityID(
            securityIDComboBox.getSelectedItem().toString());
}

private void startButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    String symbol = symbolComboBox.getSelectedItem().toString();
    String securityID =
securityIDComboBox.getSelectedItem().toString();
    int rate = (int)rateSlider.getValue();
    if ( rate == 0 ) rate = 1;
    Integer delay = 60000 / rate;
    FIXimulator.getApplication().startIOIsender( delay, symbol,
securityID );
}

private void sliderChanged(javax.swing.event.ChangeEvent evt) {
   if (!rateSlider.getValueIsAdjusting()) {
        int rate = (int)rateSlider.getValue();
        if ( rate == 0 ) rate = 1;
        Integer newDelay = 60000 / rate;
        FIXimulator.getApplication().setNewDelay(newDelay);
    }
}

private void stopButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    FIXimulator.getApplication().stopIOIsender();
}

private void singleIOIButtonActionPerformed(java.awt.event.ActionEvent
evt) {
    dialogIOI = new IOI();
    dialogIOI.setType("NEW");
    ioiDialog.setTitle("Add IOI...");
    ioiDialogID.setText(dialogIOI.getID());
    ioiDialogShares.setValue(0);
    ioiDialogSymbol.setText("");
    ioiDialogSecurityID.setText("");
    ioiDialogPrice.setValue(0.0);
    ioiDialog.pack();
    ioiDialog.setVisible(true);
}
```

```
private void
startExecutorButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int delay = 1;
    if ( this.executorDelay.getSelectedItem().toString().equals("10
ms"))
        delay = 10;
    if ( this.executorDelay.getSelectedItem().toString().equals("100
ms"))
        delay = 100;
    if ( this.executorDelay.getSelectedItem().toString().equals("1
second"))
        delay = 1000;
    if ( this.executorDelay.getSelectedItem().toString().equals("5
seconds"))
        delay = 5000;
    int partials = (int)partialsSlider.getValue();
    if ( partials == 0 ) partials = 1;
    FIXimulator.getApplication().startExecutor(delay, partials);
}

private void
stopExecutorButtonActionPerformed(java.awt.event.ActionEvent evt) {
    FIXimulator.getApplication().stopExecutor();
}

private void ioiDialogCancelActionPerformed(java.awt.event.ActionEvent
evt) {
    ioiDialog.dispose();
}

private void ioiDialogOKActionPerformed(java.awt.event.ActionEvent evt)
{
    // Set fields
    dialogIOI.setSide(ioiDialogSide.getSelectedItem().toString());
    dialogIOI.setQuantity(Integer.parseInt(ioiDialogShares.getText()));
    dialogIOI.setSymbol(ioiDialogSymbol.getText());
    dialogIOI.setSecurityID(ioiDialogSecurityID.getText());

dialogIOI.setIDSource(ioiDialogIDSource.getSelectedItem().toString());
    dialogIOI.setPrice(Double.parseDouble(ioiDialogPrice.getText()));

dialogIOI.setNatural(ioiDialogNatural.getSelectedItem().toString());
    FIXimulator.getApplication().sendIOI(dialogIOI);
    ioiDialog.dispose();
}

private void cancelIOIButtonActionPerformed(java.awt.event.ActionEvent
evt) {
    int row = ioiTable.getSelectedRow();
    // if there is a row selected
    if ( row != -1 ) {
        row = ioiTable.convertRowIndexToModel(row);
        IOI ioi = FIXimulator.getApplication().getIOIs().getIOI(row);
        IOI cancelIOI = ioi.clone();
        cancelIOI.setType("CANCEL");
        FIXimulator.getApplication().sendIOI(cancelIOI);
    }
```

```
}

private void replaceIOIButtonActionPerformed(java.awt.event.ActionEvent
evt) {
    int row = ioiTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        ioiDialog.setTitle("Replace IOI...");
        row = ioiTable.convertRowIndexToModel(row);
        IOI ioi = FIXimulator.getApplication().getIOIs().getIOI(row);
        dialogIOI = ioi.clone();
        dialogIOI.setType("REPLACE");

        ioiDialogID.setText(dialogIOI.getID());
        String side = dialogIOI.getSide();
        if (side.equals("BUY")) ioiDialogSide.setSelectedIndex(0);
        if (side.equals("SELL")) ioiDialogSide.setSelectedIndex(1);
        if (side.equals("UNDISCLOSED"))
ioiDialogSide.setSelectedIndex(2);
        ioiDialogShares.setValue(dialogIOI.getQuantity());
        ioiDialogSymbol.setText(dialogIOI.getSymbol());
        ioiDialogSecurityID.setText(dialogIOI.getSecurityID());
        String idSource = dialogIOI.getIDSource();
        if (idSource.equals("CUSIP"))
ioiDialogIDSource.setSelectedIndex(0);
        if (idSource.equals("SEDOL"))
ioiDialogIDSource.setSelectedIndex(1);
        if (idSource.equals("RIC"))
ioiDialogIDSource.setSelectedIndex(2);
        if (idSource.equals("TICKER"))
ioiDialogIDSource.setSelectedIndex(3);
        if (idSource.equals("OTHER"))
ioiDialogIDSource.setSelectedIndex(4);
        ioiDialogPrice.setValue(dialogIOI.getPrice());
        ioiDialogNatural.setSelectedIndex(0);
        if (dialogIOI.getNatural().equals("NO"))
            ioiDialogNatural.setSelectedIndex(1);
        ioiDialog.pack();
        ioiDialog.setVisible(true);
    }

}

private void
acknowledgeButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int row = orderTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = orderTable.convertRowIndexToModel(row);
        Order order =
FIXimulator.getApplication().getOrders().getOrder(row);
        if ( order.getStatus().equals("Received") ||
             order.getStatus().equals("Pending New")) {
            FIXimulator.getApplication().acknowledge(order);
        } else {
            System.out.println(
                    "Order in status \"" + order.getStatus() + "\" " +
                    "cannot be acknowledged...");
        }
    }
}

private void partialsSliderChanged(javax.swing.event.ChangeEvent evt) {
   if (!this.partialsSlider.getValueIsAdjusting()) {
        int partials = (int)partialsSlider.getValue();
        if ( partials == 0 ) partials = 1;
        System.out.println("The number of partials was changed to: " +
partials);
        FIXimulator.getApplication().setNewExecutorPartials(partials);
    }
}

private void
executionBustButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int row = executionTable.getSelectedRow();
    // if there is a row selected
    if ( row != -1 ) {
        row = executionTable.convertRowIndexToModel(row);
        Execution execution =

FIXimulator.getApplication().getExecutions().getExecution(row);
        if ( execution.getExecType().equals("Fill") ||
             execution.getExecType().equals("Partial fill")) {
            FIXimulator.getApplication().bust(execution);
        } else {
            System.out.println(
                    "\"" + execution.getExecType() + "\" " +
                    "executions cannot be busted...");
        }

    }
}

private void executorDelayActionPerformed(java.awt.event.ActionEvent
evt) {
    int delay = 1;
    String value = executorDelay.getSelectedItem().toString();
    if (value.equals("10 ms")) delay = 10;
    if (value.equals("100 ms")) delay = 100;
    if (value.equals("1 second")) delay = 1000;
    if (value.equals("5 seconds")) delay = 5000;
    FIXimulator.getApplication().setNewExecutorDelay(delay);
}

private void autoReplaceActionPerformed(java.awt.event.ActionEvent evt)
{
    FIXimulator.getApplication().getSettings()
        .setBool("FIXimulatorAutoReplace",
        autoReplace.isSelected());
}

private void
autoPendingCancelActionPerformed(java.awt.event.ActionEvent evt) {
    FIXimulator.getApplication().getSettings()
```

```
        .setBool("FIXimulatorAutoPendingCancel",
        autoPendingCancel.isSelected());
}

private void autoAcknowledgeActionPerformed(java.awt.event.ActionEvent
evt) {
    FIXimulator.getApplication().getSettings()
        .setBool("FIXimulatorAutoAcknowledge",
        autoAcknowledge.isSelected());
}

private void
autoPendingReplaceActionPerformed(java.awt.event.ActionEvent evt) {
    FIXimulator.getApplication().getSettings()
        .setBool("FIXimulatorAutoPendingReplace",
        autoPendingReplace.isSelected());
}

private void autoCancelActionPerformed(java.awt.event.ActionEvent evt)
{
    FIXimulator.getApplication().getSettings()
        .setBool("FIXimulatorAutoCancel",
        autoCancel.isSelected());
}

private void
saveSettingsButtonActionPerformed(java.awt.event.ActionEvent evt) {
    FIXimulator.getApplication().saveSettings();
}

private void exitMenuItemActionPerformed(java.awt.event.ActionEvent
evt) {
System.exit(0);
}

    @SuppressWarnings("static-access")
private void
loadInstrumentMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
int result = instrumentFileChooser.showOpenDialog(this);
    if (result == instrumentFileChooser.APPROVE_OPTION) {
        File file = instrumentFileChooser.getSelectedFile();
        FIXimulator.getInstruments().reloadInstrumentSet(file);
    } else {
        System.out.println("User cancelled loading file...");
    }
}

private void aboutMenuItemActionPerformed(java.awt.event.ActionEvent
evt) {
aboutDialog.pack();
    aboutDialog.setVisible(true);
}

private void
pricePrecisionComboActionPerformed(java.awt.event.ActionEvent evt) {
    FIXimulator.getApplication().getSettings()
            .setLong("FIXimulatorPricePrecision",
Long.valueOf(pricePrecisionCombo.getSelectedItem().toString()));
}

private void
sendOnBehalfOfCompIDActionPerformed(java.awt.event.ActionEvent evt) {
    FIXimulator.getApplication().getSettings()
        .setBool("FIXimulatorSendOnBehalfOfCompID",
        sendOnBehalfOfCompID.isSelected());
}

private void
sendOnBehalfOfSubIDActionPerformed(java.awt.event.ActionEvent evt) {
    FIXimulator.getApplication().getSettings()
        .setBool("FIXimulatorSendOnBehalfOfSubID",
        sendOnBehalfOfSubID.isSelected());
}

private void
cachedObjectsComboActionPerformed(java.awt.event.ActionEvent evt) {
    FIXimulator.getApplication().getSettings()
            .setLong("FIXimulatorCachedObjects",

Long.valueOf(cachedObjectsCombo.getSelectedItem().toString()));
}

private void
showSettingsButtonActionPerformed(java.awt.event.ActionEvent evt) {

System.out.println(FIXimulator.getApplication().getSettings().toString(
));
}

private void
orderRejectButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int row = orderTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = orderTable.convertRowIndexToModel(row);
        Order order =
FIXimulator.getApplication().getOrders().getOrder(row);
        if ( order.getStatus().equals("Received") ||
             order.getStatus().equals("Pending New")) {
            FIXimulator.getApplication().reject(order);
        } else {
            System.out.println(
                    "Order in status \"" + order.getStatus() + "\" " +
                    "cannot be rejected...");
        }
    }
}

private void cancelButtonActionPerformed(java.awt.event.ActionEvent
evt) {
    int row = orderTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = orderTable.convertRowIndexToModel(row);
        Order order =
FIXimulator.getApplication().getOrders().getOrder(row);
        FIXimulator.getApplication().cancel(order);
    }
}

private void dfdButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int row = orderTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = orderTable.convertRowIndexToModel(row);
        Order order =
FIXimulator.getApplication().getOrders().getOrder(row);
        FIXimulator.getApplication().dfd(order);
    }
}

private void
cancelPendingButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int row = orderTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = orderTable.convertRowIndexToModel(row);
        Order order =
FIXimulator.getApplication().getOrders().getOrder(row);
        if ( order.isReceivedCancel() ) {
            FIXimulator.getApplication().pendingCancel(order);
        } else {
            System.out.println(
                    "Order is not in a valid status for pending
cancel");
        }
    }
}

private void
cancelAcceptButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int row = orderTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = orderTable.convertRowIndexToModel(row);
        Order order =
FIXimulator.getApplication().getOrders().getOrder(row);
        FIXimulator.getApplication().cancel(order);
    }
}

private void
replacePendingButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int row = orderTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = orderTable.convertRowIndexToModel(row);
        Order order =
FIXimulator.getApplication().getOrders().getOrder(row);
        if ( order.isReceivedReplace() ) {
            FIXimulator.getApplication().pendingReplace(order);
        } else {
            System.out.println(
                    "Order is not in a valid status for pending
replace");
        }
    }
}

private void
replaceAcceptButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int row = orderTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = orderTable.convertRowIndexToModel(row);
        Order order =
FIXimulator.getApplication().getOrders().getOrder(row);
        if ( order.isReceivedReplace() ||
                order.getStatus().equals("Pending Replace") ) {
            FIXimulator.getApplication().replace(order);
        } else {
            System.out.println(
                    "Order is not in a valid status to replace");
        }
    }
}

private void
cancelRejectButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int row = orderTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = orderTable.convertRowIndexToModel(row);
        Order order =
FIXimulator.getApplication().getOrders().getOrder(row);
        if ( order.isReceivedCancel() ||
                order.getStatus().equals("Pending Cancel") ) {
            FIXimulator.getApplication().rejectCancelReplace(order,
true);
        } else {
            System.out.println(
                    "Order is not in a valid status to reject
cancellation");
        }
    }
}

private void
replaceRejectButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int row = orderTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = orderTable.convertRowIndexToModel(row);
        Order order =
FIXimulator.getApplication().getOrders().getOrder(row);
        if ( order.isReceivedReplace() ||
                order.getStatus().equals("Pending Replace") ) {
```

```
            FIXimulator.getApplication().rejectCancelReplace(order,
false);
        } else {
            System.out.println(
                    "Order is not in a valid status to reject replace
request");
        }
    }
}

private void
executionDialogOKActionPerformed(java.awt.event.ActionEvent evt) {
    dialogExecution.setLastShares(
            Integer.parseInt(executionDialogShares.getText()));
    dialogExecution.setLastPx(
            Double.parseDouble(executionDialogPrice.getText()));
    String refID = dialogExecution.getRefID();
    // New execution
    if ( refID == null ) {
        FIXimulator.getApplication().execute(dialogExecution);
    // Correction
    } else {
        FIXimulator.getApplication().correct(dialogExecution);
    }
    executionDialog.dispose();
}

private void
executionDialogCancelActionPerformed(java.awt.event.ActionEvent evt) {
    executionDialog.dispose();
}

private void executeButtonActionPerformed(java.awt.event.ActionEvent
evt) {
    int row = orderTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = orderTable.convertRowIndexToModel(row);
        Order order =
FIXimulator.getApplication().getOrders().getOrder(row);
        dialogExecution = new Execution(order);
        executionDialogShares.setValue(0);
        executionDialogPrice.setValue(0.0);
        executionDialog.pack();
        executionDialog.setVisible(true);
    }

}

private void
executionCorrectButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int row = executionTable.getSelectedRow();
    // if no rows are selected
    if ( row != -1 ) {
        row = executionTable.convertRowIndexToModel(row);
        Execution execution =
FIXimulator.getApplication().getExecutions().getExecution(row);
        if ( execution.getExecType().equals("Fill") ||
             execution.getExecType().equals("Partial fill")) {
            dialogExecution = execution.clone();
            executionDialogShares.setValue(execution.getLastShares());
            executionDialogPrice.setValue(execution.getLastPx());
            executionDialog.pack();
            executionDialog.setVisible(true);
        } else {
            System.out.println(
                    "\"" + execution.getExecType() + "\" " +
                    "executions cannot be corrected...");
        }
    }
}

private void logToFileActionPerformed(java.awt.event.ActionEvent evt) {
    FIXimulator.getApplication().getSettings()
        .setBool("FIXimulatorLogToFile",
        sendOnBehalfOfSubID.isSelected());
}

private void logToDBActionPerformed(java.awt.event.ActionEvent evt) {
    FIXimulator.getApplication().getSettings()
        .setBool("FIXimulatorLogToDB",
        sendOnBehalfOfSubID.isSelected());
}

private void
customQueryRunButtonActionPerformed(java.awt.event.ActionEvent evt) {
    QueryTableModel qtm = (QueryTableModel) reportTable.getModel();
    qtm.setQuery(queryText.getText().trim());
}

private void
cannedQueryRunButtonActionPerformed(java.awt.event.ActionEvent evt) {
    QueryTableModel qtm = (QueryTableModel) reportTable.getModel();
    String can = cannedQueryCombo.getSelectedItem().toString();
    String symbol =
querySymbolText.getText().toString().trim().toLowerCase();
    String query = "";

    if (can.equals("Show last 10 IOIs..."))
        query = "select id,text from messages_log "
                + "where text like '%35=6%' order by id desc limit
10;";
    if (can.equals("Show last 10 orders..."))
        query = "select id,text from messages_log "
                + "where text like '%35=D%' order by id desc limit
10;";
    if (can.equals("Show last 10 executions..."))
                query = "select id,text from messages_log "
                + "where text like '%35=8%' order by id desc limit
10;";
    if (can.equals("Show all IOIs where Symbol(55) is..."))
        query = "select id,text from messages_log "
```

```
                + "where text like '%35=6%' and lower(text) like '%55="
                + symbol + "%';";
        if (can.equals("Show all orders where Symbol(55) is..."))
            query = "select id,text from messages_log "
                + "where text like '%35=D%' and lower(text) like '%55="
                + symbol + "%';";
        if (can.equals("Show all executions where Symbol(55) is..."))
                query = "select id,text from messages_log "
                + "where text like '%35=8%' and lower(text) like '%55="
                + symbol + "%';";
        if (can.equals("Show all activity where Symbol(55) is..."))
                query = "select id,text from messages_log "
                + "where lower(text) like '%55=" + symbol + "%';";

        qtm.setQuery(query);

        if (reportTable.getColumnCount() > 1) {

    reportTable.getColumnModel().getColumn(0).setPreferredWidth(50);

    reportTable.getColumnModel().getColumn(1).setPreferredWidth(1000);
        }
    }

        public static void main(String args[]) {
            java.awt.EventQueue.invokeLater(new Runnable() {
                public void run() {
                    fiximulator = new FIXimulator();
                    fiximulator.start();
                    new FIXimulatorFrame().setVisible(true);
                }
            });
        }

        // Variables declaration - do not modify
        private javax.swing.JDialog aboutDialog;
        private javax.swing.JLabel aboutDialogLabel;
        private javax.swing.JMenuItem aboutMenuItem;
        private javax.swing.JPanel aboutPanel;
        private javax.swing.JButton acknowledgeButton;
        private javax.swing.JPanel appSettingsPanel;
        private javax.swing.JCheckBox autoAcknowledge;
        private javax.swing.JCheckBox autoCancel;
        private javax.swing.JPanel autoExecutePanel;
        private javax.swing.JPanel autoIOIPanel;
        private javax.swing.JCheckBox autoPendingCancel;
        private javax.swing.JCheckBox autoPendingReplace;
        private javax.swing.JCheckBox autoReplace;
        private javax.swing.JPanel autoResponsePanel;
        private javax.swing.JComboBox cachedObjectsCombo;
        private javax.swing.JLabel cachedObjectsLabel;
        private javax.swing.JButton cancelAcceptButton;
        private javax.swing.JButton cancelButton;
        private javax.swing.JButton cancelIOIButton;
        private javax.swing.JButton cancelPendingButton;
        private javax.swing.JButton cancelRejectButton;
        private javax.swing.JSeparator cancelSeparator;
```

```
private javax.swing.JComboBox cannedQueryCombo;
private javax.swing.JButton cannedQueryRunButton;
private javax.swing.JLabel clientConnectedLabel;
private javax.swing.JButton customQueryRunButton;
private javax.swing.JLabel delayLabel;
private javax.swing.JButton dfdButton;
private javax.swing.JButton executeButton;
private javax.swing.JPanel executionActionPanel;
private javax.swing.JButton executionBustButton;
private javax.swing.JButton executionCorrectButton;
private javax.swing.JDialog executionDialog;
private javax.swing.JButton executionDialogCancel;
private javax.swing.JButton executionDialogOK;
private javax.swing.JFormattedTextField executionDialogPrice;
private javax.swing.JFormattedTextField executionDialogShares;
private javax.swing.JPanel executionPanel;
private javax.swing.JLabel executionPriceLabel;
private javax.swing.JScrollPane executionScrollPane;
private javax.swing.JLabel executionSharesLabel;
private javax.swing.JTable executionTable;
private javax.swing.JComboBox executorDelay;
private javax.swing.JLabel executorRunningLabel;
private javax.swing.JMenuItem exitMenuItem;
private javax.swing.JMenu fileMenu;
private javax.swing.JMenu helpMenu;
private javax.swing.JFileChooser instrumentFileChooser;
private javax.swing.JMenu instrumentMenu;
private javax.swing.JPanel instrumentPanel;
private javax.swing.JScrollPane instrumentScrollPane;
private javax.swing.JTable instrumentTable;
private javax.swing.JDialog ioiDialog;
private javax.swing.JButton ioiDialogCancel;
private javax.swing.JLabel ioiDialogID;
private javax.swing.JComboBox ioiDialogIDSource;
private javax.swing.JComboBox ioiDialogNatural;
private javax.swing.JButton ioiDialogOK;
private javax.swing.JFormattedTextField ioiDialogPrice;
private javax.swing.JTextField ioiDialogSecurityID;
private javax.swing.JFormattedTextField ioiDialogShares;
private javax.swing.JComboBox ioiDialogSide;
private javax.swing.JTextField ioiDialogSymbol;
private javax.swing.JLabel ioiIDLabel;
private javax.swing.JLabel ioiIDSourceLabel;
private javax.swing.JLabel ioiNaturalLabel;
private javax.swing.JPanel ioiPanel;
private javax.swing.JLabel ioiPriceLabel;
private javax.swing.JScrollPane ioiScrollPane;
private javax.swing.JLabel ioiSecurityIDLabel;
private javax.swing.JLabel ioiSenderRunningLabel;
private javax.swing.JLabel ioiSharesLabel;
private javax.swing.JLabel ioiSideLabel;
private javax.swing.JLabel ioiSliderLabel;
private javax.swing.JLabel ioiSymbolLabel;
private javax.swing.JTable ioiTable;
private javax.swing.JMenuItem loadInstrumentMenuItem;
private javax.swing.JPanel loadPanel;
private javax.swing.JCheckBox logToDB;
private javax.swing.JCheckBox logToFile;
private javax.swing.JLabel logToFileLabel;
private javax.swing.JMenuBar mainMenuBar;
private javax.swing.JTabbedPane mainTabbedPane;
private javax.swing.JPanel manualIOIPanel;
private javax.swing.JPanel messageDetailPanel;
private javax.swing.JScrollPane messageDetailScrollPane;
private javax.swing.JTable messageDetailTable;
private javax.swing.JPanel messagePanel;
private javax.swing.JScrollPane messageScrollPane;
private javax.swing.JTable messageTable;
private javax.swing.JSeparator oboCompIDSeparator;
private javax.swing.JSeparator oboCompIDSeparator1;
private javax.swing.JButton okButton;
private javax.swing.JPanel orderActionPanel;
private javax.swing.JPanel orderPanel;
private javax.swing.JButton orderRejectButton;
private javax.swing.JScrollPane orderScrollPane;
private javax.swing.JTable orderTable;
private javax.swing.JLabel partialsLabel;
private javax.swing.JLabel partialsNumber;
private javax.swing.JSlider partialsSlider;
private javax.swing.JComboBox pricePrecisionCombo;
private javax.swing.JLabel pricePrecisionLabel;
private javax.swing.JLabel queryLabel;
private javax.swing.JLabel querySymbolLabel;
private javax.swing.JTextField querySymbolText;
private javax.swing.JTextField queryText;
private javax.swing.JLabel rateDisplayLable;
private javax.swing.JSlider rateSlider;
private javax.swing.JButton replaceAcceptButton;
private javax.swing.JButton replaceIOIButton;
private javax.swing.JButton replacePendingButton;
private javax.swing.JButton replaceRejectButton;
private javax.swing.JSeparator replaceSeparator;
private javax.swing.JPanel reportActionPanel;
private javax.swing.JPanel reportPanel;
private javax.swing.JScrollPane reportScrollPane;
private javax.swing.JTable reportTable;
private javax.swing.JButton saveSettingsButton;
private javax.swing.JComboBox securityIDComboBox;
private javax.swing.JLabel securityIDLabel;
private javax.swing.JCheckBox sendOnBehalfOfCompID;
private javax.swing.JCheckBox sendOnBehalfOfSubID;
private javax.swing.JPanel settingsPanel;
private javax.swing.JButton showSettingsButton;
private javax.swing.JButton singleIOIButton;
private javax.swing.JButton startButton;
private javax.swing.JButton startExecutorButton;
private javax.swing.JPanel statusBarPanel;
private javax.swing.JButton stopButton;
private javax.swing.JButton stopExecutorButton;
private javax.swing.JComboBox symbolComboBox;
private javax.swing.JLabel symbolLabel;
private org.jdesktop.beansbinding.BindingGroup bindingGroup;
// End of variables declaration
```

```
}
```

IOICellRenderer.java

```
/*
 * File     : IOICellRenderer.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This renderer is used on the JTable that displays IOI
 *            messages and colors the messages based on their types.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.ui;

import java.awt.Color;
import java.awt.Component;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;

public class IOICellRenderer  extends DefaultTableCellRenderer {

    @Override
    public Component getTableCellRendererComponent(JTable table, Object
value,
            boolean isSelected, boolean hasFocus, int row, int column)
{

        int myRow = table.convertRowIndexToModel(row);
        Component component =
super.getTableCellRendererComponent(table, value,
                                          isSelected, hasFocus, myRow,
column);
        String type = (String) ((IOITableModel)table.getModel())
                .getValueAt(myRow, 1);
        if (type.equals("NEW")) {
            component.setForeground(Color.BLACK);
        }
        if (type.equals("CANCEL")) {
            component.setForeground(Color.RED);
        }
        if (type.equals("REPLACE")) {
            component.setForeground(Color.BLUE);
        }
        return component;
    }
}
```

IOITableModel.java

```
/*
 * File     : IOITableModel.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is the TableModel for the IOI Table.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.ui;

import javax.swing.table.AbstractTableModel;
import edu.harvard.fas.zfeledy.fiximulator.core.FIXimulator;
import edu.harvard.fas.zfeledy.fiximulator.core.IOI;
import edu.harvard.fas.zfeledy.fiximulator.core.IOIset;

public class IOITableModel extends AbstractTableModel {
    private static IOIset iois =
FIXimulator.getApplication().getIOIs();
    private static String[] columns =
        {"ID", "Type", "Side", "Shares", "Symbol", "Price",
         "SecurityID", "IDSource", "Natural", "RefID"};


    public IOITableModel(){
        FIXimulator.getApplication().getIOIs().addCallback(this);
    }

    public int getColumnCount() {
        return columns.length;
    }

    @Override
    public String getColumnName(int column) {
        return columns[column];
    }

    @Override
    public Class getColumnClass(int column) {
        if (column == 3) return Double.class;
        if (column == 5) return Double.class;
        return String.class;
    }

    public int getRowCount() {
        return iois.getCount();
    }

    public Object getValueAt(int row, int column) {
        IOI ioi = iois.getIOI(row);
        if (column == 0) return ioi.getID();
        if (column == 1) return ioi.getType();
        if (column == 2) return ioi.getSide();
        if (column == 3) return ioi.getQuantity();
        if (column == 4) return ioi.getSymbol();
        if (column == 5) return ioi.getPrice();
        if (column == 6) return ioi.getSecurityID();
        if (column == 7) return ioi.getIDSource();
        if (column == 8) return ioi.getNatural();
        if (column == 9) return ioi.getRefID();
        return new Object();
    }
    public void update() {
        fireTableDataChanged();
    }
}
```

InstrumentTableModel.java

```
/*
 * File     : InstrumentTableModel.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is the TableModel for the Instrument Table.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.ui;

import javax.swing.table.AbstractTableModel;
import edu.harvard.fas.zfeledy.fiximulator.core.FIXimulator;
import edu.harvard.fas.zfeledy.fiximulator.core.Instrument;
import edu.harvard.fas.zfeledy.fiximulator.core.InstrumentSet;

public class InstrumentTableModel extends AbstractTableModel {
    private static InstrumentSet instruments =
FIXimulator.getInstruments();
    private static String[] columns =
        {"Ticker", "Name", "Sedol", "RIC", "Cusip", "Price"};

    public InstrumentTableModel(){
        FIXimulator.getInstruments().addCallback(this);
    }

    public int getColumnCount() {
        return columns.length;
    }

    @Override
    public String getColumnName(int column) {
        return columns[column];
    }

    @Override
    public Class getColumnClass(int column) {
        if (column == 5) return Double.class;
        return String.class;
    }

    public int getRowCount() {
        return instruments.getCount();
    }

    public Object getValueAt(int row, int column) {
        Instrument instrument = instruments.getInstrument(row);
        if (column == 0) return instrument.getTicker();
        if (column == 1) return instrument.getName();
        if (column == 2) return instrument.getSedol();
        if (column == 3) return instrument.getRIC();
        if (column == 4) return instrument.getCusip();
        if (column == 5) return Double.valueOf(instrument.getPrice());
        return new Object();
    }

    public void update() {
        fireTableDataChanged();
    }
}
```

MessageDetailTableModel.java

```
/*
 * File     : MessageDetailTableModel.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is the TableModel for the Message Details
 *            Table.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.ui;

import java.util.ArrayList;
import java.util.List;
import javax.swing.JTable;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.table.AbstractTableModel;
import edu.harvard.fas.zfeledy.fiximulator.core.FIXimulator;
import edu.harvard.fas.zfeledy.fiximulator.core.LogMessage;
import edu.harvard.fas.zfeledy.fiximulator.core.LogMessageSet;
import edu.harvard.fas.zfeledy.fiximulator.util.LogField;

public class MessageDetailTableModel extends AbstractTableModel
        implements ListSelectionListener {
    private static LogMessageSet messages =
FIXimulator.getMessageSet();
    private JTable messageTable = null;
    private ArrayList<LogField> fields = new ArrayList<LogField>();
    private static String[] columns =
        {"Field", "Tag", "Value", "Value Name", "Required", "Section"};

    public MessageDetailTableModel(JTable messageTable){
        this.messageTable = messageTable;

messageTable.getSelectionModel().addListSelectionListener(this);
    }

    public int getColumnCount() {
        return columns.length;
    }
    @Override
    public String getColumnName(int column) {
        return columns[column];
    }

    @Override
    public Class getColumnClass(int column) {
        if (column == 1) return Integer.class;
        return String.class;
    }

    public int getRowCount() {
        return fields.size();
    }

    public Object getValueAt( int row, int column ) {
        LogField logField = fields.get( row );
        if (column == 0) return logField.getFieldName();
        if (column == 1) return logField.getTag();
        if (column == 2) return logField.getValue();
        if (column == 3) return logField.getFieldValueName();
        if (column == 4) return (logField.isRequired() ? "Yes" : "No");
        if (column == 5) {
            if (logField.isHeaderField()) return "Header";
            if (logField.isTrailerField()) return "Trailer";
            return "Body";
        }
        return null;
    }

    public void updateMessageDetailsTable(LogMessage message) {
        LogMessage logMessage = message;
        List<LogField> logFields = logMessage.getLogFields();
        fields.clear();

        for (LogField logField : logFields) {
            fields.add(logField);
        }

        fireTableDataChanged();
    }

    public void valueChanged(ListSelectionEvent selection) {
        if (!selection.getValueIsAdjusting()) {
            int row = messageTable.getSelectedRow();
            // if the first row is selected when it gets purged
            if ( row != -1 ) {
                row = messageTable.convertRowIndexToModel(row);
                LogMessage msg = messages.getMessage( row );
                updateMessageDetailsTable(msg);
            }
        }
    }
}
```

MessageTableModel.java

```
/*
 * File     : MessageTableModel.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is the TableModel for the Message Table.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.ui;

import javax.swing.table.AbstractTableModel;
import edu.harvard.fas.zfeledy.fiximulator.core.FIXimulator;
import edu.harvard.fas.zfeledy.fiximulator.core.LogMessage;
import edu.harvard.fas.zfeledy.fiximulator.core.LogMessageSet;
import quickfix.field.converter.UtcTimestampConverter;

public class MessageTableModel extends AbstractTableModel {
    private static LogMessageSet messages =
FIXimulator.getMessageSet();
    private static String[] columns =
        {"#", "Direction", "SendingTime", "Type", "Message"};

    public MessageTableModel(){
        messages.addCallback(this);
    }

    public int getColumnCount() {
        return columns.length;
    }

    @Override
    public String getColumnName(int column) {
        return columns[column];
    }

    @Override
    public Class getColumnClass(int column) {
        if (column == 0) return Integer.class;
        return String.class;
    }

    public int getRowCount() {
        return messages.getCount();
    }

    public Object getValueAt( int row, int column ) {
        LogMessage msg = messages.getMessage( row );
        if ( column == 0 ) return msg.getMessageIndex();
        if ( column == 1 ) return (msg.isIncoming() ? "incoming" :
"outgoing");
        if ( column == 2 ) return
UtcTimestampConverter.convert(msg.getSendingTime(),true);
        if ( column == 3 ) return msg.getMessageTypeName();
        if ( column == 4 ) return msg.getRawMessage();
        return new Object();
    }

    public void update() {
        fireTableDataChanged();
    }
}
```

OrderTableModel.java

```
/*
 * File     : OrderTableModel.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is the TableModel for the Order Table.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.ui;

import javax.swing.table.AbstractTableModel;
import edu.harvard.fas.zfeledy.fiximulator.core.FIXimulator;
import edu.harvard.fas.zfeledy.fiximulator.core.Order;
import edu.harvard.fas.zfeledy.fiximulator.core.OrderSet;

public class OrderTableModel extends AbstractTableModel {
    private static OrderSet orders =
FIXimulator.getApplication().getOrders();
    private static String[] columns =
        {"ID", "Status", "Side", "Quantity", "Symbol", "Type", "Limit",
"TIF",
         "Executed", "Open", "AvgPx", "ClOrdID", "OrigClOrdID"};


    public OrderTableModel(){
        FIXimulator.getApplication().getOrders().addCallback(this);
    }

    public int getColumnCount() {
        return columns.length;
    }

    @Override
    public String getColumnName(int column) {
        return columns[column];
    }

    @Override
    public Class getColumnClass(int column) {
        if (column == 3) return Double.class;
        if (column == 6) return Double.class;
        if (column == 8) return Double.class;
        if (column == 9) return Double.class;
        if (column == 10) return Double.class;
        return String.class;
    }

    public int getRowCount() {
        return orders.getCount();
    }

    public Object getValueAt(int row, int column) {
        Order order = orders.getOrder(row);
        if (column == 0) return order.getID();
        if (column == 1) return order.getStatus();
        if (column == 2) return order.getSide();
        if (column == 3) return order.getQuantity();
        if (column == 4) return order.getSymbol();
        if (column == 5) return order.getType();
        if (column == 6) return order.getLimit();
        if (column == 7) return order.getTif();
        if (column == 8) return order.getExecuted();
        if (column == 9) return order.getOpen();
        if (column == 10) return order.getAvgPx();
        if (column == 11) return order.getClientID();
        if (column == 12) return order.getOrigClientID();
        return new Object();
    }

    public void update() {
        fireTableDataChanged();
    }
}
```

QueryTableModel.java

```
/*
 * File     : QueryTableModel.java
 *
 * Author   : Zoltan Feledy
 *
 * Contents : This class is the TableModel for the SQL queries for
 *            reporting.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.ui;

import java.sql.*;
import edu.harvard.fas.zfeledy.fiximulator.core.FIXimulator;
import java.util.Vector;
import javax.swing.table.AbstractTableModel;

class QueryTableModel extends AbstractTableModel {
    Vector results = new Vector();
    private static String[] columns = {"Results"};
    Connection connection;
    Statement statement;
    String url;
    String driver;
    String user;
    String pass;

    public QueryTableModel() {
        try {
            url = FIXimulator.getApplication().getSettings()
                    .getString("JdbcURL");
            driver = FIXimulator.getApplication().getSettings()
                    .getString( "JdbcDriver");
            user = FIXimulator.getApplication().getSettings()
                    .getString("JdbcUser");
            pass = FIXimulator.getApplication().getSettings()
                    .getString("JdbcPassword");
        } catch (Exception e) {}
    }

    public int getRowCount() {
        return results.size();
    }

    public int getColumnCount() {
        return columns.length;
    }

    @Override
    public String getColumnName(int column) {
        return columns[column];
    }

    public Object getValueAt(int row, int column) {
        return ((String[])results.elementAt(row))[column];
    }

    public void setQuery(String query) {
        results = new Vector();

        try {
            Class.forName(driver).newInstance();
            connection = DriverManager.getConnection(url, user, pass);
        }
        catch(Exception e) {
            System.out.println("Could not initialize the database.");
            e.printStackTrace();
        }

        try {
            statement = connection.createStatement();
            ResultSet rs = statement.executeQuery(query);
            ResultSetMetaData meta = (ResultSetMetaData)
rs.getMetaData();
            int fields = meta.getColumnCount();
            columns = new String[fields];
            for ( int i=0; i < fields; i++) {
                columns[i] = meta.getColumnName(i+1);
            }
            while (rs.next()) {
                String[] record = new String[fields];
                for (int i=0; i < fields; i++) {
                    record[i] = rs.getString(i + 1);
                }
                results.addElement(record);
            }
            statement.close();
            rs.close();
            fireTableChanged(null);
        } catch(Exception e) {
            results = new Vector();
            e.printStackTrace();
        }

        if (connection != null) {
            try {
                connection.close ();
            } catch (Exception e) {}
        }
    }
}
```

FIXMessageHelper.java

```
/*
 * File     : FIXMessageHelper.java
 *
 * Author   : Brian M. Coyner
 *
 * Contents : This class is a helper class used by Log4FIX for
 *            handling message details.  It was adapted for the needs
 *            of FIXimulator by Zoltan Feledy
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.util;
/*
 * The Log4FIX Software License
 * Copyright (c) 2006 - 2007 opentradingsolutions.org  All rights
reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. Neither the name of the product (Log4FIX), nor
opentradingsolutions.org,
 *    nor the names of its contributors may be used to endorse or
promote
```

```
 *    products derived from this software without specific prior
written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL OPENTRADINGSOLUTIONS.ORG OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
import java.util.Date;

import quickfix.FieldConvertError;
import quickfix.field.converter.UtcTimestampConverter;

/**
 * @author Brian M. Coyner
 */
public class FIXMessageHelper {

    public static String getTargetCompId(String rawMessage, char
delimeter) {
        int beginIndex = rawMessage.indexOf("56=") + 3;
        int endIndex = rawMessage.indexOf(delimeter, beginIndex);
        return rawMessage.substring(beginIndex, endIndex);
    }

    public static String getSenderCompId(String rawMessage, char
delimeter) {
        int beginIndex = rawMessage.indexOf("49=") + 3;
        int endIndex = rawMessage.indexOf(delimeter, beginIndex);
        return rawMessage.substring(beginIndex, endIndex);
    }

    public static String getMessageType(String rawMessage, char
delimeter) {
        int beginIndex = rawMessage.indexOf("35=");
        if (beginIndex == -1) {
            return null;
        }

        beginIndex += 3;

        int endIndex = rawMessage.indexOf(delimeter, beginIndex);
        return rawMessage.substring(beginIndex, endIndex);
    }

    public static Date getSendingTime(String rawMessage, char
delimeter)
            throws FieldConvertError {
```

```
        int beginIndex = rawMessage.indexOf("52=");
        if (beginIndex == -1) {
            return null;
        }

        beginIndex += 3;

        int endIndex = rawMessage.indexOf(delimeter, beginIndex);
        return UtcTimestampConverter.convert(
                rawMessage.substring(beginIndex, endIndex));
    }
}
```

LogField.java

```
/*
 * File     : LogField.java
 *
 * Author   : Brian M. Coyner
 *
 * Contents : This class is an enhanced Log Field used by the Log4FIX
 *            project.  Adapted for the needs of FIXimulator
 *            by Zoltan Feledy.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.util;

/*
 * The Log4FIX Software License
 * Copyright (c) 2006 - 2007 opentradingsolutions.org  All rights
reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. Neither the name of the product (Log4FIX), nor
opentradingsolutions.org,
 *    nor the names of its contributors may be used to endorse or
promote
 *    products derived from this software without specific prior
written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL OPENTRADINGSOLUTIONS.ORG OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import quickfix.DataDictionary;
import quickfix.Field;
import quickfix.FieldType;
import quickfix.field.MsgType;

/**
 * Represents a single QuickFIX field. This object provides extra
information
 * about the field in the context of its message. The QuickFIX
 * <code>DataDictionary</code> is used to provide information about
whether or
 * not the field is a header or trailer field, if it is a required
field, etc.
 *
 * @author Brian M. Coyner
 */
public class LogField {

    private Field field;
    private FieldType fieldType;
    private String fieldName;
    private String fieldValueName;
    private boolean required;
    private boolean header;
    private boolean trailer;
    private List<LogGroup> groups;

    private DataDictionary dictionary;

    public static LogField createLogField(MsgType messageType, Field
field,
            DataDictionary dictionary) {
        return new LogField(messageType, field, dictionary);
    }

    /**
     * @param messageType what message the field is part of.
     * @param field the actual field we are wrapping.
     * @param dictionary dictionary used to look up field information.
     */
    protected LogField(
            MsgType messageType, Field field, DataDictionary
dictionary) {
        this.dictionary = dictionary;
        this.field = field;

        final String messageTypeString = messageType.getValue();
        final int fieldTag = field.getTag();

        fieldType = dictionary.getFieldTypeEnum(fieldTag);
        fieldName = dictionary.getFieldName(fieldTag);
        fieldValueName = dictionary.getValueName(fieldTag,
                field.getObject().toString());
        required =
getDataDictionary().isRequiredField(messageTypeString,
                fieldTag);
        header = getDataDictionary().isHeaderField(fieldTag);
        if (!header) {
            trailer = getDataDictionary().isTrailerField(fieldTag);
        }
    }

    public Iterator<LogField> group() {
        return null;
    }

    public Field getField() {
        return field;
    }

    public FieldType getFieldType() {
        return fieldType;
    }

    public int getTag() {
        return field.getTag();
    }

    public Object getValue() {
        return field.getObject();
    }

    public String getFieldName() {
        return fieldName;
    }

    public String getFieldValueName() {
        return fieldValueName;
    }

    public boolean isRequired() {
        return required;
    }

    public boolean isHeaderField() {
        return header;
    }
    public boolean isTrailerField() {
        return trailer;
    }

    public boolean isRepeatingGroup() {
        return groups != null;
    }

    /**
     * @return true if this this field is not a header field or a
trailer field.
     */
    public boolean isBodyField() {
        return !isHeaderField() || !isTrailerField();
    }

    public DataDictionary getDataDictionary() {
        return dictionary;
    }

    public void addGroup(LogGroup group) {
        if (groups == null) {
            groups = new ArrayList<LogGroup>();
        }

        groups.add(group);
    }

    public List<LogGroup> getGroups() {
        return groups;
    }
}
```

LogGroup.java

```
/*
 * File     : LogGroup.java
 *
 * Author   : Brian M. Coyner
 *
 * Contents : This class is an enhanced Log Group used by the Log4FIX
 *            project for handling repeating groups.
 *            Adapted for the needs of FIXimulator by Zoltan Feledy.
 *
 */

package edu.harvard.fas.zfeledy.fiximulator.util;
/*
 * The Log4FIX Software License
 * Copyright (c) 2006 - 2007 opentradingsolutions.org  All rights
reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. Neither the name of the product (Log4FIX), nor
opentradingsolutions.org,
 *    nor the names of its contributors may be used to endorse or
promote
 *    products derived from this software without specific prior
written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL OPENTRADINGSOLUTIONS.ORG OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

import java.util.ArrayList;
import java.util.List;

import quickfix.DataDictionary;
import quickfix.Field;
import quickfix.field.MsgType;

/**
 * @author Brian M. Coyner
 */
public class LogGroup extends LogField {

    private List<LogField> fields;

    public LogGroup(MsgType messageType, Field field,
            DataDictionary dictionary) {
        super(messageType, field, dictionary);
        fields = new ArrayList<LogField>();
    }

    public void addField(LogField logField) {
        fields.add(logField);
    }

    public List<LogField> getFields() {
        return fields;
    }
}
```

XML Files

instruments_small.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<instruments>
   <instrument name="TXU CORP" ticker="TXU" sedol="2885700" ric="TXU.N"
cusip="873168108" price="58.56"/>
   <instrument name="INTL BUSINESS MACHINES CORP" ticker="IBM"
sedol="2005973" ric="IBM.N" cusip="459200101" price="92.07"/>
   <instrument name="YAHOO! INC" ticker="YHOO" sedol="2986539"
ric="YHOO.OQ" cusip="984332106" price="32.09"/>
   <instrument name="EBAY INC" ticker="EBAY" sedol="2293819"
ric="EBAY.OQ" cusip="278642103" price="32.59"/>
   <instrument name="AMAZON.COM INC" ticker="AMZN" sedol="2000019"
ric="AMZN.OQ" cusip="023135106" price="34.15"/>
</instruments>
```

Text Files

FIXimulator.cfg

```
[DEFAULT]
FIXimulatorLogToFile=N
FIXimulatorAutoPendingCancel=N
FIXimulatorAutoPendingReplace=N
FIXimulatorLogToDB=N
StartTime=00:00:00
JdbcUser=fiximulator
FIXimulatorSendOnBehalfOfCompID=N
JdbcPassword=fiximulator
FIXimulatorAutoAcknowledge=N
FIXimulatorAutoCancel=N
FIXimulatorAutoReplace=N
FIXimulatorPricePrecision=4
FIXimulatorSendOnBehalfOfSubID=N
SocketAcceptPort=9878
RefreshMessageStoreAtLogon=Y
BeginString=FIX.4.2
HeartBtInt=30
EndTime=00:00:00
JdbcDriver=com.mysql.jdbc.Driver
ConnectionType=acceptor
DataDictionary=FIX42.xml
FileStorePath=data
FIXimulatorCachedObjects=50
JdbcURL=jdbc:mysql://localhost:3306/quickfix
[SESSION]
OnBehalfOfSubID=DESK
FileLogPath=logs
TargetCompID=BANZAI
SenderCompID=FIXIMULATOR
OnBehalfOfCompID=BROKER
```

SQL Files

fiximulator_user.sql

```
grant all privileges on quickfix.*
to 'fiximulator'@'localhost'
identified by 'fiximulator';
```