FINSWITCH

PROJECT WORK DONE AT
ELIND COMPUTERS (P) LIMITED,
BANGALORE.

P- 813

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE AWARD OF THE DEGREE OF
M.Sc [APPLIED SCIENCE] SOFTWARE ENGINEERING
OF BHARATHIAR UNIVERSITY, COIMBATORE.

SUBMITTED BY

R.VISHNU PRAKASH
REG NO. **9937S0100**

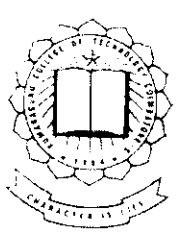
UNDER THE GUIDANCE OF

External Guide

Mr. B.Kingshuk
ELIND Computers,
Bangalore.

Internal guide

Mrs. Devaki
Dept. Of CSE.,
Coimbatore – 6



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE – 641 006
MAY 2002 – AUG 2002

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

(Affiliated to Bharathiar University)
COIMBATORE – 641 002
SEPTEMBER – 2002

CERTIFICATE

This is to certify that the project entitled

FINSWITCH

DONE BY

R.VISHNU PRAKASH
REG NO. **9937S0100**

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF
M.Sc [Applied science] SOFTWARE ENGINEERING
OF BHARATHIYAR UNIVERSITY

Professor and HOD 27/9

Internal Guide

Submitted to University Examination held on 27/9/2002

Internal Examiner 27/9/2002

External Examiner

elind

Elind Computers Private Limited

4032, 100 Feet Road, Indiranagar, Bangalore -560 038. INDIA Tel: +91-80-5216767 Fax : +91-80-5216761

September 20, 2002

CERTIFICATE

This is to certify that Vishnu Prakash (VII Semester MSc, Kumaraguru College of Technology, Coimbatore) was involved in the project titled " FinSwitch" at Elind Computers Private Limited.

Sincerely,

Suman Joshi
Manager - HR

DECLARATION

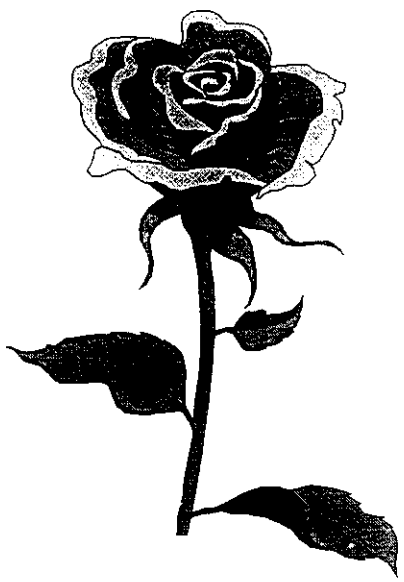
I hereby declare that the project entitled "**FINSwitch**" submitted to Bharathiar University as the project work of Master Of Software Engineering Degree, is a record of original work done by me under the supervision and guidance of **Mr.Kingshuk Bandyapadhyay**, *Senior Member Technical, Elind Computers, Bangalore* and **Mrs.Devaki** *Senior Lecturer, Department of Computer Science and Engineering, Kumaraguru College of Technology, Coimbatore* and this project work has not found the basis for the award of any Degree/Diploma/Associate ship/Fellowship or similar title to any candidate of any University.

Place: Coimbatore
Date:

(Vishnu Prakash.R)
Reg.No.9937s0100

Countersigned By

(Internal Guide)



DEDICATED TO MY EVER LOVING PARENTS

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

I express my profound respect and sincere gratitude to **Dr. K.K.Padmanaban PhD**, *Principle, Kumaraguru College of Technology, Coimbatore*, for providing me an opportunity to undergo the Msc [APPLIED SCIENCE SOFTWARE ENGINEERING] Course and thereby this project work also.

I record my sincere thanks to **Dr. S.Thangaswamy PhD**, *Head of the Department, Computer Science and Engineering, Kumaraguru College of Technology*, for allowing me to take up the project at *Elind Computers Private Limited, Bangalore.*

It's my privilege to express my deep sense of gratitude and profound thanks to **Mr.Mangal Das Shetty, Chief Executive Officer**, *Elind Computers private Limited, Bangalore,* for having allowed me to carry out the project at their esteemed organization.

Gratitude will find least meaning without a mention to my guide **Mrs. S.Devaki BE MS. Assistant professor**, *Department of Computer Science and Engineering, Kumaraguru College of Technology*, who has guided me with her valuable suggestions and constant motivations during my project work.

Words are boundless for me to express my deep sense of gratitude and profound thanks to **Mr.Kingshuk Bandyapadhyay**, *Senior Member Technical*, and all my associates at *Elind Computers, Bangalore* for all their kind guidance and encouragement towards my project work. I am happy that I was able to give shape to their novel ideas to an extent.

Finally, this acknowledgement will not achieve its complete form if I don't remember my parent's sacrifices. Without their constant moral support, motivations and kind encouragements, I could not have channelised my career in the field of Computer Science.

Vishnu Prakash.R

PROJECT ABSTRACT

PROJECT ABSTRACT

The project entitled "FINSwitch" is developed for Elind Computers Private Limited.

FINSwitch, a powerful and extensible, multi-platform and multi-protocol messaging platform for securities trading networks developed by Elind. FINSwitch is an integral component of Elind's Straight Through Processing (STP) product offerings that offers products across the securities trade cycle

FINSwitch is a financial messaging solution that serves as a messaging layer for financial institutions across the securities chain (buy-side, sell side, execution destinations and post-trade intermediaries). FINSwitch plays the role of a messaging, translation and routing of data.

FINSwitch supports the translation and routing of bi-directional message flows wherein messages emanating in application proprietary formats are converted to FIX/FIXML and messages based on FIX/FIXML are converted to the pertinent application API. The direction (routing) of message flows intelligently, for both inbound and outbound messages and between internal applications is a core capability of the product.

CONTENTS

CONTENTS

PREFACE

1 PREFACE

This document, entitled, FINSwitch is a technical reference guide that accompanies the Elind FINSwitch software.

This section details the document conventions, target audience and how to go about reading it.

1.1 INTENDED AUDIENCE

The audience for this guide is system integrators, developers and user interface designers responsible for writing adapters for FINSwitch to connect to host applications.

The readers of this guide are expected to have good understanding of Java Programming language and FIX Protocol.

1.2 PURPOSE

This document presents a high-level functional and technical overview of FINSwitch with an objective to acquaint readers with the product's functionality and capabilities.

The document is aimed at the technology decision-makers at financial institutions evaluating financial messaging solutions and other personnel involved with their organization's STP initiative/project office.

This guide is designed to provide programming information related to FINSwitch interfaces.

1.3 ORGANIZATION PROFILE

Elind delivers mission critical software products & solutions to the securities & related financial services industry. Its products automate the entire transaction chain by providing order routing, order management, order matching, clearing & settlement &

depository solutions. These products and solutions increase operational efficiency and minimize risks.

The Professional Services Group within Elind specializes in business consultancy and delivery of technology projects to the securities industry and related financial services sector.

With the implementation of two major overseas exchange contracts with Abuja Stock Exchange, Nigeria and TradingLab, Milan, Elind has made an entry into the international arena.

Today, Elind's products cover the entire transaction chain to suit the various needs of stock exchanges, commodities exchanges, brokerage houses, clearing houses, depositories and new generation marketplaces like ATS, ECNs, B2B exchanges and financial portals.

Elind is headquartered in Bangalore with sales and support offices in Mumbai, New York and London.

PRODUCTS

Elind's products and solutions cater to the securities industry and related financial services sector covering stock exchanges, ATS, ECNs, brokerage houses, primary markets as well as negotiated trading. Built on robust technology, the products can be easily interfaced with customers' existing systems. The products are built on open platforms and are highly scalable and reliable.

Elind's product offerings:	
STRIDE™	Trading solution for equity and debt instruments
Infinet C&S	Integrated clearing & settlement system for equity, debt and derivative instruments
ATS STRIDE™	An exchange system for Alternative

	Trading Systems (ATS)
COMM STRIDE™	An exchange solution for commodities marketplaces
TradePort™	Multi-exchange order routing and risk management system
TradeiPort™	Web channel server which enables Internet trading
IPO STRIDE™	Web-based IPO (Initial Public Offering) system with multiple allocation models
FINSwitch™	Muti-protocol message engine

1.4 PROGRAMMING ENVIRONMENT

1.4.1 Hardware & software Configuration

System Requirements

The minimum system requirements, needed to install FINSwitch, are listed below.

Hardware Requirements

The recommended hardware for FINSwitch is as follows:

- CPU - Pentium III 500 MHz or above
- Memory - 128 MB RAM
- 20 GB HDD
- 32X CD ROM Drive
- Disk Storage – 20 MB free space

Software Requirements

Solaris 8 (Sparc)-

- Java run time environment - Sun JRE 1.3.1_03
- Databases - Oracle 8i using thin driver or HSQL Database 1.6.1
- JMS - JMS server shipped with Sun Micro system's J2EE reference implementation 1.3

[By combining Java technology with enterprise messaging, the Java Messaging Service (JMS) API provides a new, powerful tool for solving enterprise-computing problems. The JMS API improves programmer productivity by defining a common set of messaging concepts and programming strategies that will be supported by all JMS technology-compliant messaging systems.

Windows 2000 and Windows NT 4.0 service pack 6 –

- Java run time environment - Sun JRE 1.3.1_03
- Databases - Oracle 8i using thin driver, MSSQL Server 7.0/MSSQL Server 2000

JMS - JMS server shipped with Sun Micro system's J2EE reference implementation.

1.4.2 Description of the Software

Never before has any new programming language attracted so much attention and becomes so popularly so quickly. In the first year of its existence, java took the web by storm and became its adapted programming language. Since then, java has become the language of choice for developing both applications and applets, and is used for both business and consumer software development. The java phenomenon has captivated the imaginations of programmers around the world and is leading a way towards the next era of distributed application development. Java's appeal lies in its simplicity, its familiarity, and careful selection of features that it includes and excludes. A government comity or a clique of academics did not design java. Its shares the spirit with c more than any syntactical similarities. It is programming language that was designed by programmers for programmers.

The reason that so much attention has been paid to java is summarized in the following list. Java allows the developer to do

- ✓ Write robust and developer to do

- ✓ Built an application on almost any platform and run that application on any other supported platform without recompiling the code
- ✓ Distribute the application over the network in a secured fashion.
- ✓ Java is Object Oriented
- ✓ Java is platform independent
- ✓ Java is safe
- ✓ Java is reliable

SECURITY

Java provides an excellent security towards the web applications by providing a *"Firewall"* between networked application and your computer. The ability to download applets with confidence that no harm will be done and that no security will be breached is considered by many to be the single most important aspect of java.

PORTABILITY

Many types of computers and operating systems are in use throughout the world and many are connected to Internet. For programs to be dynamically downloaded to all various types of platforms connected to Internet, some means of generating portable executable code is needed. Java's solution to these two problems is both elegant and efficient.

JAVA'S MAGIC: THE BYTECODE

The key that allows Java to solve both the security and the portability problems just described is that the output of a Java compiler is not executable code. Rather, it is bytecode. Bytecode is a highly optimized set of instructions designed to be executed by the java run-time system, which is called the *Java Virtual Machine (JVM)*. That is, in its standard form, the JVM is an interpreter for bytecode. Translating a java program into

bytecode helps makes it much easier to run a program in a wide variety of environments. The reason is straightforward: only the JVM is to be implemented for each platform. Once the run-time package exists for a given system, any java program can run on it.

ARCHITECTURAL NEUTRAL

Java language follows the principle of *"write once; run anywhere, any time, forever"*. To a great extend this goal was accomplished.

DISTRIBUTED

Java is designed for the distributed environment of the Internet, because it handles TCP/IP protocols. The concept of *Remote Method Invocation (RMI)* brings unparalleled level of abstraction to client/server programming.

DYNAMIC

Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run-time. This makes it possible to dynamically link code in a safe and expedient manner.

MULTI-THREADED

Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously. The Java run-time system comes with an elegant yet sophisticated solution for multiprocess synchronization that enables you to construct smoothly running interactive systems. Java's easy-to-use approach to multithreaded allows you to think about the specific behavior of your program, not the multitasking subsystem.

NETWORKING

A network is like an electric socket. Network sockets are TCP/IP packets and IP address. Internet protocol is a low level routing protocol that breaks data into small

packets and sends them to address across a network. Transport control protocol is a higher-level protocol that manages a robustly string together these packets, sorting and retransmitting then as necessary.

1.5 ORGANIZATION OF THE DOCUMENT

The document is organized in the following manner

Table: Organization of the Document

Chapter	*Description*
Introduction to FIX	Describes FIX protocol
Introduction to FINSwitch	Introduces you to FINSwitch and provides information about the features supported in this release
FINSwitch Architecture	Introduces you to FINSwitch architecture
Programming interfaces to FINSwitch	Deals with programming interfaces to FINSwitch
FINSwitch Library	Deals with how FINSwitch Library allows the developer to develop FIX applications using a programming interface.
Sample deployment	Deals with some sample deployment scenarios
Test Cases	Testing of Administrative Messages

1.5.1 TEXT CONVENTIONS

The document conventions followed in this manual are listed below:

Table: Text Conventions

Font	*Field*
Italic	Table Title and Image Captions
Bold	Titles
Normal	Notes

1.5.2 ABBREVIATIONS AND NOTATIONS

This section describes the various Abbreviations and Notations used in the manual.

Abbreviation	*Description*
FIX	Financial Information eXchange
FIXML	Financial Information eXchange Markup Language
J2EE	Java 2 Enterprise Edition

JDK	Java Development Kit
JMS	Java Messaging Service
JRE	Java Runtime Environment
JMX	Java Management extension
JDBC	Java Database Connectivity
SSL	Secure Sockets Layer
OMS	Online Marketing Strategies
ECN	Explicit Congestion Notification
IOI	Indication Of Interest
SOAP	Simple Object Access Protocol
XML	eXtended Markup Language
LDAP	Light Weight Directory Application
ATS	Alternative Trading Systems

INTRODUCTION TO FIX

2 INTRODUCTION TO FIX

The Financial Information Exchange (FIX) protocol is a message standard developed to facilitate the electronic exchange of information related to securities transactions. It is intended for use between trading partners wishing to automate their pre-trade, trade, post-trade and settlement instruction message flows.

FIX has evolved over a period of time from supporting US domestic equity trading (version 3.0) to cross-border Collective Investment Vehicles (CIVs), Derivatives, Fixed Income, and Foreign Exchange trading (version 4.3).

FIX is now used by a variety of firms and vendors. It has become the inter-firm messaging protocol of choice in the pre-trade activities involving Institutional Investors/Fund Managers, Brokers, Exchanges, ECN's and ATS's. FIX has helped market participants reduce the clutter of unnecessary telephone calls and scraps of paper, and facilitates targeting high quality information to specific individuals.

The FIX protocol specification is maintained by the FIX Technical Committee, which receives its direction from the International Steering Committees, the Global Steering Committee, and the various Working Groups comprised of industry participants such as fund managers, brokers, exchanges, and vendors.

The FIX Website, http://www.fixprotocol.org, serves as the central repository and authority for all specification documents, committee calendars, discussion forums, presentations, and everything FIX.

2.1 A BRIEF LOOK AT THE FIX PROTOCOL

The different elements of the FIX protocol are:

- Message Format
- Communication/Transport Layer
- Session Layer
- Application Layer

2.1.1 Message Format

- The general format of a FIX message is a standard header followed by the message body and terminated with a standard trailer.
- FIX protocol currently exists in two syntaxes

<Tag>=<value>syntax

Each message is constructed of a stream of <tag>=<value> fields with a field delimiter between fields in the stream. The non-graphic, ASCII "SOH" ("Start of Header" #001,hex:0x01, referred to in this document as <SOH> is used as field delimiter.

FIXML syntax

- FIXML uses XML as its vocabulary for creating FIX messages.
- Focuses primarily on the FIX Application Layer and does not concern itself with the session Layer.
- Can be encapsulated within the FIX Session Protocol or within another protocol like SOAP.

- Apart from a few exceptions, fields can be in any order.
- Fields can be part of a repeating data group. A repeating group encapsulates one or more repeating instances of the group of fields defined for the group, including nested repeating groups.
- Fields can belong to one of many defined types. E.g., int, float, char, Boolean, data, etc.
- Each message defined in the protocol satisfies a business function and is made up of a collection of optional and mandatory fields.
- Messages are either session related or application related.

P-513

KCT LIBRARY Coimbatore

2.1.2 Communication/Transport Layer

FIX neither demands a single type of carrier (e.g., it will work with leased lines, frame relay, Internet, etc), nor a specific security protocol. It leaves many of these decisions to the individual firms that are using it. It would be pertinent to mention that TCP/IP is the most widely implemented communications option in the FIX domain.

2.1.3 Session Layer

The protocol is defined at two levels: session and application. The session level is concerned with the delivery of data. A FIX session is defined as a bi-directional stream of ordered messages between two parties within a continuous sequence number series. Some characteristics are:

- A unique sequence number per session, for ensuring ordered delivery identifies all FIX messages.
- All FIX messages have a body length and checksum field for ensuring data integrity.
- Fields can be encrypted and also present as plain text. Useful for validation and verification.
- Automatic recovery is built into the protocol
- Supported by messages like Heartbeat, Logon, Logout, Reject, Resend

2.1.4 Application Layer

The exchange of business related information is accomplished through the passing of application messages. The different application messages defined in the protocol fall into the following categories:

- **Pre-Trade** - These are messages that are typically communicated prior to the placement of an order. E.g. IOI, News, Quote, Market Data, etc.

- **Orders and Executions** – These are messages that are used to place or amend orders and communicate the results and status of orders. E.g. New Order Single, Execution Report (Trade), Order Cancel Request, etc.
- **Post-Trade** – These are messages that are typically communicated after the placement and successful execution of an order and prior to settlement. E.g. Allocation, Settlement Instructions, etc.

INTRODUCTION TO FINSWITCH

3 INTRODUCTION TO FINSWITCH

FINSwitch is a financial messaging solution that serves as a messaging layer for financial institutions across the securities chain (buy-side, sell side, execution destinations and post-trade intermediaries). FINSwitch plays the role of a messaging, translation and routing intermediary between a financial institution's trading/back office applications that communicate through proprietary API and systems that use FIX, FIXML protocols. These FIX/FIXML systems could be applications within an organization (order routing gateways, crossing engines etc) and external destinations (broker-dealers, investment managers, post-trade intermediaries, execution destinations etc) working on FIX/FIXML.

FINSwitch supports the translation and routing of bi-directional message flows wherein messages emanating in application proprietary formats are converted to FIX/FIXML and messages based on FIX/FIXML are converted to the pertinent application API. The direction (routing) of message flows intelligently, for both inbound and outbound messages and between internal applications is a core capability of the product.

The forthcoming releases for the product offer high protocol extensibility with support for SWIFT ISO 15022, FpML, OFX & IFX. Protocol adaptors for connectivity to the virtual matching Utilities (VMUs) - OMGEO and GSTP are also planned in the future releases.

FINSwitch is a FIX Engine written in 100% pure java programming language. It currently supports FIX protocol versions 4.2 and 4.3. This section gives an overview of the features in the current release from the following aspects:

- Supported Platforms
- Message Protocol Support
- Transport Support
- Programming Library Support

- Monitoring Support
- Extensibility

3.1 SUPPORTED PLATFORMS

The current release of FINSwitch has been tested on the following platforms:

JRE version	Operating System	Database and JDBC Driver
1.3.1_03 from Sun Microsystems	*Solaris 2.8 on Sparc hardware.*	*Oracle 8i with thin driver* *HSQL 1.6.1*
1.3.1_03 from Sun Microsystems	*Windows 2000 Service Pack 2*	*Oracle 8i (8.1.7) with thin driver* *MS SQLServer 7.0 with JDBC-ODBC driver from Sun Microsystems*
1.3.1_03 from Sun Microsystems	*Windows NT 4.0 Service Pack 6*	*Oracle 8i (8.1.7) with thin driver* *MS SQLServer 7.0 with the JDBC-ODBC driver from Sun Microsystems*

3.2 MESSAGE PROTOCOL SUPPORT

The current release of FINSwitch provides complete session level and application level message support for FIX.4.2 and FIX.4.3.

3.3 TRANSPORT PROTOCOL SUPPORT

Current transport protocol support include the following:

Simple TCP/IP based communication

SSL V3 using JSSE 1.0.2 from Sun Microsystems.

JMS 1.0.2 using Sun Microsystems reference implementation of JMS Provider.

Counterparties can communicate with FINSwitch using any of the above transport mechanisms.

3.4 PROGRAMMING LIBRARY SUPPORT

The current release of FINSwitch comes with a Java library that can be used to build FIX client and server applications.

The library provides Java classes to:

Compose complete FIX.4.2 and FIX.4.3 session and application level messages in Java.

- Validate FIX messages from a user defined XML message definition dictionary.
- Use FIX 4.2 or 4.3 session to build FIX client or server.

The rest of the document will describe in detail how to use the library along with programming examples.

3.5 MONITORING SUPPORT

FINSwitch comes with Java Management Extension (JMX) based Managed Beans (MBeans) to monitor the state of all FIX session and transport related objects within FINSwitch. The current release of FINSwitch has been tested with Sun Microsystems's reference implementation of JMX server 1.0.1. FINSwitch ships with the reference implementation's JMX server and provides a browser based monitoring screen to display the state of FIX sessions and transport related activities.

3.6 EXTENSIBILITY

To take into account the varied needs of an enterprise's requirement of integrating a FIX engine within its existing software infrastructure. FINSwitch has been developed in a highly extensible manner. Its behavior can be altered and extended by writing custom classes and hosting them within FINSwitch. The current release of FINSwitch allows altering (i.e. plugging in custom implementations of) the following aspects:

- Authentication mechanism – By writing appropriate authentication class and supplying it to FINSwitch.
- Message validation mechanism – either by extending the supplied validation mechanism or by developing new validation logic.

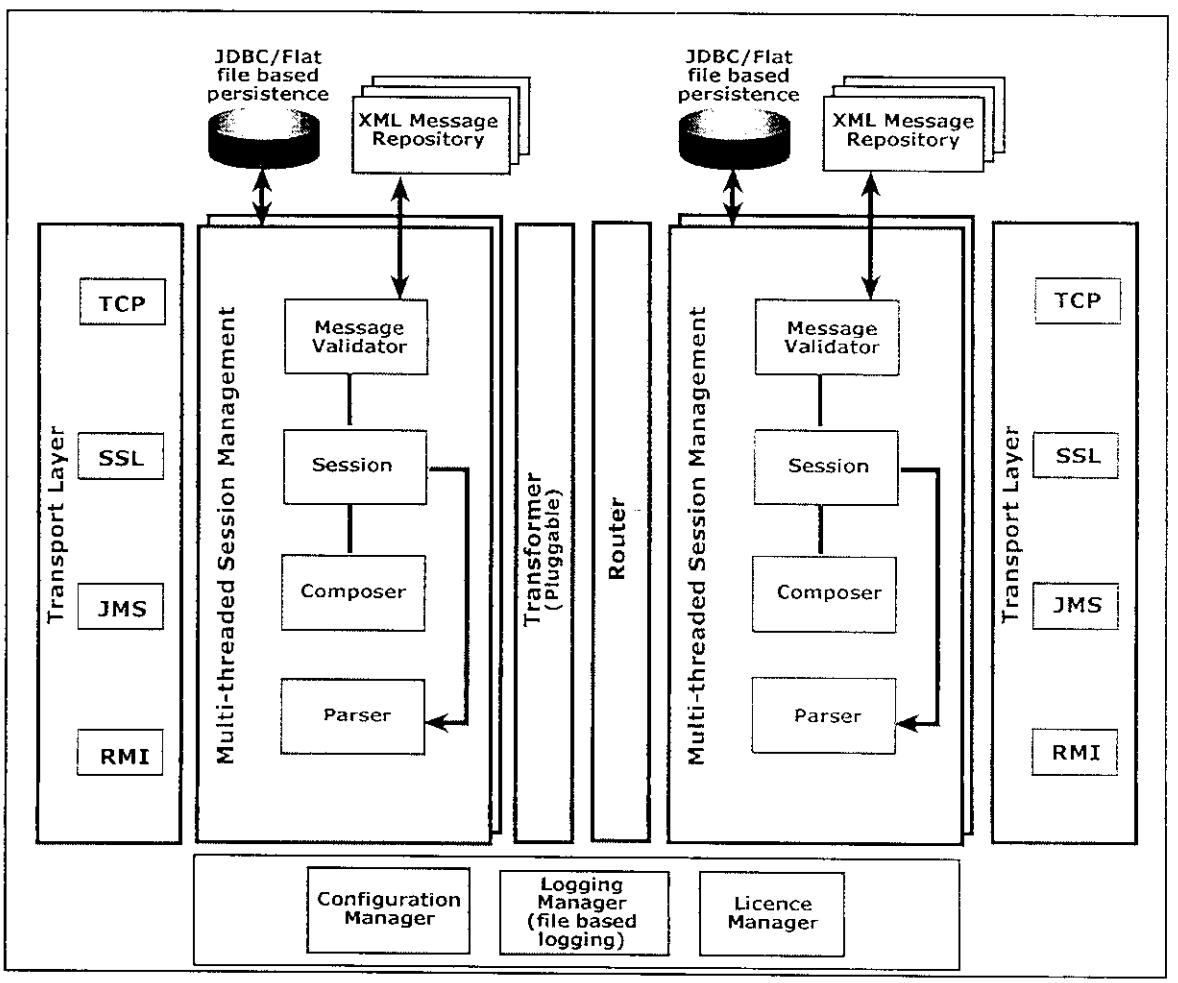
- New message support – By modifying the supplied XML based message dictionary.
- New host application protocol support – By writing a host application specific Dispatcher class and instructing FINSwitch to load it.

The rest of the document will describe how each of the above aspects can be modified.

FINSWITCH ARCHITECTURE

4 FINSWITCH ARCHITECTURE

FINSwitch architecture is based on open, industry standard Java and J2EE technologies like JMS. This ensures that FINSwitch can be integrated into existing IT infrastructure by leveraging the wide support base available for Java-based products. Most subsystems within FINSwitch provide a pluggable interface wherein custom classes can be loaded and integrated within FINSwitch easily. For example, the Security Manager module within FINSwitch, used to authenticate a user, can be replaced with a customized Security Manager that leverages the existing security infrastructure to support, say, LDAP based authentication.



Logical View

4.1 TRANSPORT LAYER

This layer provides connectivity between FINSwitch and its clients. As indicated in the above diagram the current release of FINSwitch supports TCP, SSL, JMS as transport mechanism.

The main features of the transport layer are:

4.1.1 Built-in Reconnects

The transport layer can detect a network failure and can attempt recovery. This ensures that temporary network unavailability over a particular link does not bring down the entire FINSwitch instance or disconnect all other clients.

4.1.2 Remote Management and Monitoring

The transport module provides JMX based Managed Beans that can be used to monitor and manage the system from a remote browser-based console.

4.2 SESSION MANAGEMENT LAYER

This layer provides message parsing, validation and session management ability for a particular message protocol. Multiple session management implementations (for each of the supported message protocols) can be loaded within FINSwitch. This enables support for multiple message protocols in a single FINSwitch instance. The current release of FINSwitch provides support for FIX 4.2 and 4.3. Message validation is driven by XML definition of messages available in a message dictionary. Adding support for new messages or custom tags will involve creating a new XML file containing the message definition and adding it to the repository. The main features of the layer are:

4.2.1 Pluggable interface

By developing appropriate classes, support for custom message formats and protocols used by legacy applications can be added. FINSwitch can be configured to use these classes before delivering a message to a host application

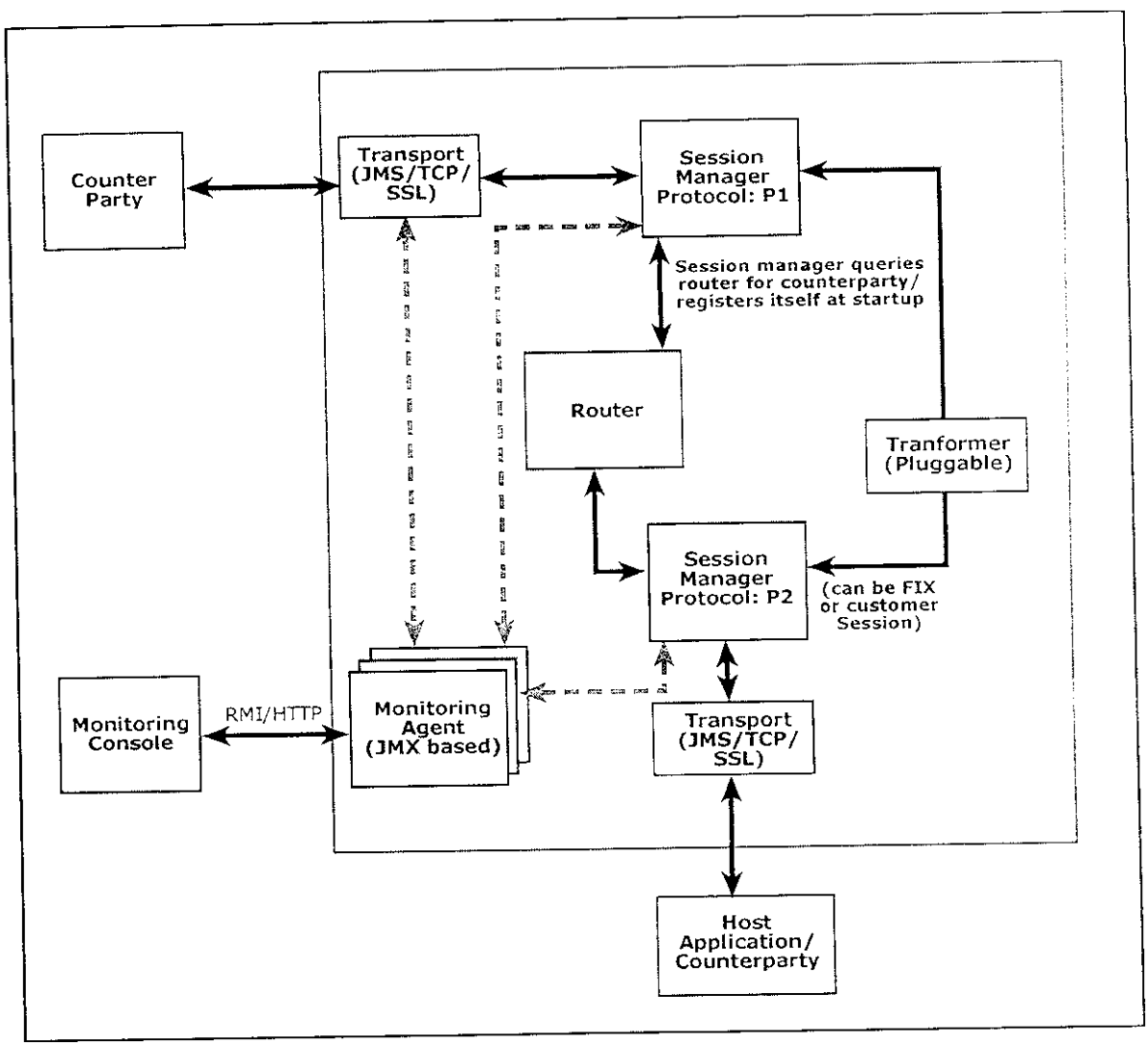
4.2.2 Remote monitoring

This subsystem provides JMX based managed beans that publish all the fix session events allowing a remote administrative console to monitor and detect any problems within the system

4.3 ROUTER

The session management layer uses the router component to find out the counterparty/host application to which an application message should be delivered. By default, FINSwitch is installed with a FIX router that can identify a destination by inspecting FIX message headers.

4.4 RUN-TIME VIEW



Run time view

The diagram above shows a run time snapshot of a single instance of FINSwitch where two counterparties are connected through FINSwitch.

The following are the steps during run time:

1. Counterparty initiates a FIX session by connecting to FINSwitch via any of the supported transport mechanisms
2. Depending on the transport end point the counterparty connecting to FINSwitch creates a particular session manager for the counterparty
3. The session manager carries out the initial handshake required by the protocol along with any other session messages that the protocol demands
4. The counterparty sends a business application message that has to be delivered to another counterparty (or host application) connected to FINSwitch.
5. The session manager finds the appropriate destination by querying the router. The appropriate destination is another session manager acting on behalf of the destination. This could be a session manager supporting the same protocol. Or it could be a custom session manager specifically written for that host
6. In case the two session managers speak different message protocols, an appropriate message transformer can be embedded in the second session to convert the counterparty message format into host specific message format
7. The message will be sent to the host application using its session manager

PROGRAMMING INTERFACES

5 PROGRAMMING INTERFACES TO FINSWITCH

5.1 DEPLOYMENT SCENARIOS

Some applicable deployment scenarios are mentioned before describing the usage of the interfaces exposed by FINSwitch:

- FINSwitch fronts the order management/trading/execution system (referred to as the Host), exchanging FIX messages with counterparties (possibly supporting multiple FIX versions). The Host system may be an OMS, an Exchange, ECN, ATS or an internal marketplace.
- The Host is a buy-side Institutional Investor, Broker/Dealer or Fund Manager who would initiate connections to multiple sell-side/trading partner destinations that provide execution systems or accessing pools of liquidity.
- The Host is a sell-side Exchange, ECN, ATS or Settlement system that would accept connections from other sell-side/buy-side/trading partner systems for providing trade executions/market data/trade settlement and other facilities.

5.2 Authenticator

The Authenticator is an interface exposed by FINSwitch for authenticating users connecting to the system.

5.2.1 Where is it used

FINSwitch comes with a default authenticator that validates a username/password with the information stored in the FINSwitch participant database. Alternatively, FINSwitch can be configured to use an object that adheres to the Authenticator interface for user authentication. The implementation can refer to an existing database (RDBMS, LDAP, etc) for this purpose.

5.2.2 Relevant packages

com.elind.finswitch.processing.Authenticator;

com.elind.finswitch.protocol.fix.exception.AuthenticationException;

5.3 FIXLoginCreator

The *FIXLoginCreator* is an interface exposed by the FINSwitch for allowing Dispatcher objects representing an Initiator session to create the Logon message to be sent to the counterparty in the counterparty-specific fashion.

5.3.1 Where is it used

The *FixLoginCreator* is used by Dispatcher objects representing an Initiator session to create the Logon message to be sent to the counterparty in the counterparty-specific fashion.

5.3.2 Relevant packages

com.elind.finswitch.processing.FixLoginCreator;
com.elind.finswitch.protocol.fix.composer.FixMessag;
com.elind.finswitch.protocol.fix.session.FixSessionInfo;
com.elind.finswitch.protocol.fix.session.FixCounterPartyInfo;

FINSWITCH LIBRARY

6. FINSWITCH LIBRARY

The FINSwitch Library is a suite of components that allows the developers to develop FIX applications using a programming interface. It hides the complexities of the session protocol from the developer so that the developer is free to concentrate on the actual business message processing.

The FINSwitch Library is composed of the following functional blocks:

- Message Composer
- Message Validator
- Session
- Monitoring

6.1 PROGRAMMING SCENARIOS

Some applicable programming scenarios for the usage of the FINSwitch Library are discussed below:

- Communicating with a trusting FIX server without going through a FIX session. This is very typical of an organization with multiple front-ends linking to a central OMS system, via secure internal network, for executing orders at multiple destinations
- A server application that serves as the gateway to an external market place via a FIX connection. Translates between multiple internal formats to FIX and vice versa, and validates all messages coming in and going out.
- A simple front-end order entry system, taking inputs from a trader, composes FIX messages (optionally validates the messages) and forwards them to an execution system through a connection to an ECN. Receives executions and forwards them to multiple back office systems for settlement

6.2 MESSAGE COMPOSER

The Message Composer encapsulates all the FIX 4.2 and 4.3 messages as Java objects and allows the user to manipulate the values of the tags using simple method calls on the object without being aware of the actual FIX tag id. It handles the formatting of the message object contents to FIX string and vice versa. It also allows addition of custom/user-defined tags into a message, as well as creation of custom messages.

6.2.1 Where is it used

The Message Composer can be used for FIX-enabling Java-based client applications like Order Entry System/Trading Workstation/etc.

6.2.2 How is it used

The Message Composer is a set of packages, each of which support the creation of messages specified as part of a particular FIX version. Currently there are two packages, which support FIX 4.2 and 4.3 messages. All FIX 4.2 and 4.3 messages are subclasses of Fix42Message and Fix43Message objects respectively.

The message objects provided are convenience classes hiding from the programmer the intricacies of a FIX message and its rules of formatting. The repeating groups within FIX messages have also been modeled as objects and hence ease programmer burden in understanding and coding these instances.

Pointers on how to use these objects:

- Create the message object (no-args constructor)
- Set the fields using the setter methods provided. Setter methods that take a java.util.Collection object as argument are used to set repeating groups within the message.
- If the message defines a repeating group, then create the appropriate repeating group object. Objects of name

Fix<version>Rg<MessageName>_<RepeatingGroupName> identify such repeating groups. For e.g., Fix42RgAllocationMessage_Exec is a repeating group present in the Fix42AllocationMessage.

- Add each repeating group instance to a java.util.Collection and use the setter method in the parent message object to set the repeating group. For e.g., use the setExec() method of the Fix42AllocationMessage to set the Fix42RgAllocation_Exec repeating group collection.
- Use the pack() method to calculate the body length and checksum for the message once all the fields have been set.
- Use the toString() method to extract the message as a FIX formatted string which can then be communicated to any FIX-aware application. There is an implicit call to pack() before the string is formatted.
- The toString() and pack() method call ensures that the checksum field is present (adds it otherwise).

In addition to using the message type specific message object (as described above), the developer may choose to use the Fix42Message/Fix43Message object itself. The developer should be aware of FIX message formatting specifics for doing so.

Additional pointers on using the Fix42Message/Fix43Message object for creating the messages:

- Create a Fix42Message/Fix43Message object with no-args constructor and use the addTag() methods to add the tag id and its value
- Create the FixMessage object passing the message type as argument.

Note:

a. The BeginString field is populated by default and is always the first field.

b. The BodyLength and CheckSum fields are checked for when pack() or toString() method is called and added if necessary.

6.2.2.1 Extending a Message

In addition to using the different message objects, there may arise a necessity to add custom tags/fields to an existing message or create custom business/application messages having both custom and existing FIX tags. The way to go about creating these messages is described below.

6.2.2.2 Custom/User-defined Tag

User-defined tags can be added to an existing message by extending the particular message composer class and adding the required field's setter and getter methods.

6.2.2.3 Custom/User-defined Message

User-defined messages can be created by extending the base message class. Depending upon the protocol version this will be either the Fix42Message or Fix43Message class. By doing so, the new message assumes the default behaviors of a FIX 4.2 or FIX 4.3 message and builds on it. All the fields encompassed by the new message become setter and getter methods.

6.2.3 Relevant packages

com.elind.finswitch.protocol.fix.composer.*;
com.elind.finswitch.protocol.fix.composer.fix42.*;
com.elind.finswitch.protocol.fix.composer.fix43.*;
com.elind.finswitch.protocol.fix.samples.composer.*;

6.3 MESSAGE VALIDATOR

The Message Validator encapsulates the validating logic for FIX 4.2 and 4.3 messages as Java objects and allows the user to influence the validation in a number of ways.

6.3.1 Where is it used

The Message Validator can be used for validating FIX messages and ensuring that the contract with the trading partner/counterparty is adhered to vis-à-vis the messages flowing between the two parties.

6.3.2 How is it used

The Message Validator is a set of packages, each of which support the validation of messages specified as part of a particular FIX version. Currently there are two packages, which support FIX 4.2 and 4.3 message validation. Both FIX 4.2 and 4.3 validators are subclasses of the FixValidator class.

There are a number of ways in which the validation of FIX messages can be influenced:

- By editing the FIX Message Definition/Tag Definition XML files (more on this can be found here)
- Extending the default validator
- Creating a custom validator

6.3.3 Relevant packages

com.elind.finswitch.protocol.fix.validator.*;
com.elind.finswitch.protocol.fix.validator.fix42.*;
com.elind.finswitch.protocol.fix.validator.fix43.*;
com.elind.finswitch.protocol.fix.samples.validator.*;

6.4 SESSION

Session encapsulates the logic of maintaining a FIX session with a counterparty. The FIX 4.2 and 4.3 Session related Java objects ensure ordered delivery (sequence numbers, Resend Requests, and Sequence Reset messages) and message integrity (body length and checksum). The Message Composer and Message Validator components are used for creation, manipulation and validation of FIX messages in accordance with the session-level rules defined in the FIX 4.2 and 4.3 specifications. The user has a reasonably fine-

grained control over the working of the session and can customize certain behavioral aspects of the component using the callbacks and interfaces provided for the same. Support for Initiator and Acceptor sessions with mechanism for defining custom authentication mechanisms are also provided.

6.4.1 Where is it used

It can be used for maintaining a session with FIX-speaking counterparties and ensuring the ordered delivery and integrity of the messages exchanged. It can be used by a buy-side counterparty to talk to a sell-side counterparty for executing orders and receiving/sending IOI's, or by a sell-side counterparty accepting FIX connections from buy-side counterparties for trade executions, market data, trade settlement, etc. Every FIX session has a one side initiating the first connection and logon, known as the Initiator, and the other side waiting for connections, known as the Acceptor.

6.4.2 How is it used

The Session package with support for FIX 4.2 and 4.3 sessions expose a callback interface that allow the embedding application to react to the callbacks in a business-specific fashion. Since the actual transport mechanism is not dictated by the session, the embedding application has more freedom in choosing an appropriate mode of transport for FIX-connectivity to other counterparties. The programming model for Initiator and Acceptor sessions is the same except for some startup activities specific to an Initiator and an Acceptor.

The different callback interfaces used by the session are described below:

a. FixSessionHandler

Allows the FixSession to communicate FIX messages with the counterparty to the session. It also provides the session with callbacks for validating logons, indicating session termination, and other queries to the embedding application. The FixSessionHandler gives the embedding application the freedom to choose an appropriate

transport satisfying the quality of service requirements, though TCP is the most widely implemented choice of transport.

b. FixRequestStore

Allows the FixSession to persist request messages to the persistent store. The embedding application can use an appropriate persistent store like an RDBMS, Flat File, etc. Persisting a request message may not be of much importance other than for keeping track of the message sequence number and hence a Flat File with logging of the sequence number and comp id's alone would suffice.

c. FixResponseStore

Allows the FixSession to persist response messages to the persistent store. The embedding application can use an appropriate persistent store like an RDBMS, Flat File, etc. All response messages need to be persisted as such to the persistent store as the session uses this store to service Resend requests from the counterparty. For reasons of transactional integrity and consistency it might be a good idea to use an RDBMS here. But, of course, if speed is the overwhelming criteria then a Flat File would serve the cause better.

d. FixValidator

Allows the FixSession to validate incoming and outgoing messages against the set of rules specified in the xml configuration files. Refer to the FINSwitch Administration Guide for more details.

e. FixApplicationHandler

Allows the FixSession to forward an FIX application message to the business system for further processing.

Other than the above-mentioned callback interfaces the session also requires a couple of information objects, viz., FixSessionInfo and FixCounterPartyInfo. The FixSessionInfo object encapsulates information about the FIX session related properties of the counterparty represented by the embedding application, including CompID, SubID, LocID, FIX Version and Heartbeat Interval. The FixCounterPartyInfo object captures the CompID, SubID, LocID and Heartbeat Interval information about the opposite party to the session.

SESSION AS AN INITIATOR

Other than capturing the information in the two information objects, it is also necessary to have an object that creates the Logon message, specific to the counterparty, to be sent immediately after a transport session is established. FINSwitch provides an interface, LoginCreator, which allows an appropriate instance to be plugged-in to FINSwitch runtime and can be used to create a FIX Logon message to be sent to the counterparty.

SESSION AS AN ACCEPTOR

While creating an Acceptor session only information about the accepting session is available, i.e., the FixSessionInfo object can be set. But, the counterparty's information will be available only when the first message has been received and validated. Also, it will be necessary for the embedding application to authenticate the user/password when a Logon is received. FINSwitch allows a custom authenticator, an instance of the Authenticator interface, to be plugged-in to the runtime.

6.4.3 Relevant packages

com.elind.finswitch.protocol.fix.session.*

com.elind.finswitch.protocol.fix.session.fix42.*

com.elind.finswitch.protocol.fix.session.fix43.*

com.elind.finswitch.protocol.fix.samples.session.*

com.elind.finswitch.processing.Authenticator

com.elind.finswitch.processing.FixLoginCreator

6.5 MONITORING

During the lifecycle of a FixSession object various events occur that change the state of the session. For example arrival of an application message causes the sequence number to increase by one, or arrival of a message with low sequence number causes the session to terminated. The library provides an interface called ***com.elind.finswitch.protocol.fix.monitor.SessionEventHandler*** (from now onwards called SessionEventHandler) that provides callback method signatures for each of the session events. An object of type SessionEventHandler represents a callback object that is interested in the session events.

6.5.1 Where is it used

For you as a FIX application programmer some or all of these events could be of interest for –

Audit trail

Since all the events are published to a SessionEventHandler along with the messages that caused the event you can use this mechanism to build your audit trail.

Debugging

Since all messages between a counterparty and your FIX application go through a FixSession you can monitor the session to debug any FIX related problems that may appear in your application.

Monitoring the state of a session

The state of the session can be monitored and if required an alert can be raised. For example a certain number of consecutive "ResendRequest " event could be a trigger for an alert or even session termination.

Event correlation

You may develop a SessionEventHandler class that accepts events from all the sessions in your system, inspects the messages and correlates them to trigger some other business process.

6.5.2 How is it used

The interface for the callback object is
com.elind.finswitch.protocol.fix.monitor.SessionEventHandler.
This interface provides callback methods for each session event. If an object of type SessionEventHandler is supplied to a FixSession, the FixSession calls the appropriate method whenever an event occurs.

To access the session events write a class that implements the interface SessionEventHandler and provides appropriate implementation for each callback method. After session creation you will have to supply an instance of this class to the session by invoking the setEventHandler method on the FixSession object. From that point onwards until the session termination all the events will be notified to the SessionEventHandler. While writing an implementation of SessionEventHandler take into account the following:

- The implementation class must be thread safe - Since several threads can act on the FixSession object to change its state, more than one thread can invoke the methods on the SessionEventHandler object that you write concurrently. Hence the monitor class must be thread safe.
- The FixSession can notify only one SessionEventHandler at any instance of time – Hence only one SessionEventHandler object can be used to monitor a given session at a time. The SessionEventHandler object that gets notified by a FixSession is the one that has been supplied to the session most recently.

6.5.3 Relevant packages

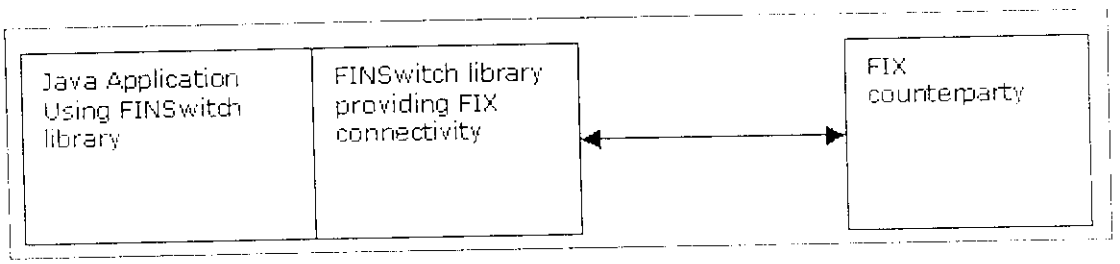
com.elind.finswitch.protocol.fix.Monitor

com.elind.finswitch.protocol.fix.Composer

com.elind.finswitch.protocol.fix.Session

7. SAMPLE DEPLOYMENT SCENARIO

7.1 SINGLE APPLICATION USING LIBRARY



7.1.1 Description

In this scenario a single java application needs to be FIX enabled so that it can communicate with FIX aware counterparty applications. The application will handle the necessary transport protocol details.

7.1.2 How to implement the scenario

You will have make use of the java library that ships with FINSwitch and use the composer, validator and session to embed FIX communication ability within your application.

8 ADMINISTRATIVE MESSAGES

8.1 MESSAGE STANDARD HEADER

Each administrative or application message is preceded by a standard header. The header identifies the message type, length, destination, sequence number, origination point and time.

Two fields help with resending messages. The PossDupFlag is set to Y when resending a message as the result of a session level event (i.e. the retransmission of a message reusing a sequence number). The PossResend is set to Y when reissuing a message with a new sequence number (e.g. resending an order). The receiving application should process these messages as follows:

PossDupFlag - if a message with this sequence number has been previously received, ignore message, if not, process normally.

PossResend – Ambiguous application level messages may be resend with the PossResend flag set. If the end user suspect a message has not reached he can send an message with PossResend flag set with same body data & a new Sequence number.

In addition checksum fields requires recalculations.

REQUIRED FIELDS IN HEADER		
Tag	*Field Name*	*Comments*
8	BeginString	To identify the FIX Version
9	BodyLength	For checking the integrity of data
35	MsgType	To identify the FIX message
49	SenderCompID	Originating Comp ID
56	TargetCompID	Destination Comp ID

34	MsgSeqNum	Unique for each FIX message
52	SendingTime	FIX Message sending time
43	PossDupFlag	Required only a message is resend
122	OrigSendingTime	Required only a message is resend

8.2 MESSAGE STANDARD TRAILER

Each message, administrative or application, is terminated by a standard trailer. The trailer is used to segregate messages and contains the three digit character representation of the Checksum value.

Standard Message Trailer

Tag	*Field Name*	*Req'd*	*Comments*
93	SignatureLength	N	Required when trailer contains signature. *Note: Not to be included within SecureData field*
89	Signature	N	*Note: Not to be included within SecureData field*
10	CheckSum	Y	*(Always unencrypted, always last field in message)*

Data Integrity

The integrity of message data content can be verified in two ways: verification of message length (body Length) and a simple checksum of characters.

Sequence Number

All FIX messages are identified by unique sequence number. Sequence Number are initialized at the start of the FIX session in the Logon Message with an initial value and incremented throughout the session.

MsgSeqNum-34(Tag Value)

Message Type

Message type is the most important field in the FIX Message that is used to identify the Type of the Message

MsgType(35)	*Message Description*
A	Logon
5	Logout
0	HeartBeat
1	TestRequest
2	ResendRequest
3	Reject
4	SequenceReset

GARBLED MESSAGE

The FIX Protocol takes the optimistic view; it presumes that a garbled message is received due to a transmission error rather than a FIX system problem. Therefore, if a Resend Request is sent the garbled message will be retransmitted correctly. If a message is not considered garbled then it is recommended that a session level Reject message be sent.

What constitutes a garbled message

- BeginString (tag #8) is not the first tag in a message or is not of the format 8=FIX.n.m.
- BodyLength (tag #9) is not the second tag in a message or does not contain the correct byte count.
- MsgType (tag #35) is not the third tag in a message.
- Checksum (tag #10) is not the last tag or contains an incorrect value.

Administrative Message	***Fields Required***
A-Logon	98- EncryptMethod
	108- HeartBeat
0-HeartBeat	112-TestRequestID(Required only while Responding to TestRequest Message)
1-TestRequest	112- TestRequestID

2-Resend Request	7- BeginSeqNo
	16-EndSeqNo
3-Reject	45-RefSeqNum
	58-Text (Optional)
4-SequenceReset-GapFill	36-NewSequenceNo
5-LogOut	Nil

8.3 LOGON MESSAGE

The logon message authenticates a user establishing a connection to a remote system. The logon message must be the first message sent by the application requesting to initiate a FIX session.

The HeartBtInt (108) field is used to declare the timeout interval for generating heartbeats (same value used by both sides). The HeartBtInt value should be agreed upon by the two firms and specified by the Logon initiator and echoed back by the Logon acceptor.

Upon receipt of a Logon message, the session acceptor will authenticate the party requesting connection and issue a Logon message as acknowledgment that the connection request has been accepted. The initiator to validate that the connection was established with the correct party can also use the acknowledgment Logon.

The session acceptor must be prepared to immediately begin processing messages after receipt of the Logon. The session initiator can choose to begin transmission of FIX messages before receipt of the confirmation Logon, however it is recommended that normal message delivery wait until after the return Logon is received to accommodate encryption key negotiation.

Logon

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = A
98	EncryptMethod	Y	(*Always unencrypted*)
108	HeartBtInt	Y	Note same value used by both sides
95	RawDataLength	N	Required for some authentication methods
96	RawData	N	Required for some authentication methods
	Standard Trailer	Y	

8.4 HEARTBEAT MESSAGE

The Heartbeat monitors the status of the communication link and identifies when the last of a string of messages was not received.

When either end of a FIX connection has not sent any data for [HeartBtInt] seconds, it will transmit a Heartbeat message. When either end of the connection has not received any data for (HeartBtInt + "some reasonable transmission time") seconds, it will transmit a Test Request message. If there is still no Heartbeat message received after (HeartBtInt + "some reasonable transmission time") seconds then the connection should be considered lost and corrective action be initiated. If HeartBtInt is set to zero then no regular heartbeat messages will be generated. Note that a test request message can still be sent independent of the value of the HeartBtInt, which will force a Heartbeat message.

Heartbeats issued as the result of Test Request must contain the TestReqID transmitted in the Test Request message. This is useful to verify that the Heartbeat is the result of the Test Request and not as the result of a regular timeout.

Heartbeat

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 0
112	TestReqID	N	Required when the heartbeat is the result of a Test Request message.
	Standard Trailer	Y	

8.5 TESTREQUEST MESSAGE

The test request message forces a heartbeat from the opposing application. The test request message checks sequence numbers or verifies communication line status. The opposite application responds to the Test Request with a Heartbeat containing the TestReqID.

The TestReqID verifies that the opposite application is generating the heartbeat as the result of Test Request and not a normal timeout. The opposite application includes the TestReqID in the resulting Heartbeat. Any string can be used as the TestReqID (one suggestion is to use a timestamp string).

Test Request

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 1
112	TestReqID	Y	
	Standard Trailer	Y	

8.6 RESEND REQUEST MESSAGE

The resend request is sent by the receiving application to initiate the retransmission of messages. This function is utilized if a sequence number gap is detected, if the receiving application lost a message, or as a function of the initialization process.

The resend request can be used to request a single message, a range of messages or all messages subsequent to a particular message.

- To request a single message: BeginSeqNo = EndSeqNo
- To request a range of messages: BeginSeqNo = first message of range, EndSeqNo = last message of range
- To request all messages subsequent to a particular message: BeginSeqNo = first message of range, EndSeqNo = 0 (represents infinity).

Resend Request

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 2
7	BeginSeqNo	Y	
16	EndSeqNo	Y	
	Standard Trailer	Y	

8.7 REJECT

The reject message should be issued when a message is received but cannot be properly processed due to a session-level rule violation. An example of when a reject may be appropriate would be the receipt of a message with invalid basic data (e.g. MsgType=&) which successfully passes de-encryption, CheckSum and BodyLength checks. As a rule, messages should be forwarded to the trading application for business level rejections whenever possible.

Rejected messages should be logged and the incoming sequence number incremented.

Note: The receiving application should disregard any message that is garbled, cannot be parsed or fails a data integrity check. Processing of the next valid FIX message will cause detection of a sequence gap and a Resend Request will be generated.

Generation and receipt of a Reject message indicates a serious error that may be the result of faulty logic in either the sending or receiving application.

If the sending application chooses to retransmit the rejected message, it should be assigned a new sequence number and sent with PossResend=Y.

Scenarios for session-level Reject:

SessionRejectReason
0 = Invalid tag number
1 = Required tag missing
2 = Tag not defined for this message type
3 = Undefined Tag
4 = Tag specified without a value
5 = Value is incorrect (out of range) for this tag
6 = Incorrect data format for value
7 = Decryption problem
8 = Signature problem
9 = CompID problem
10 = SendingTime accuracy problem
11 = Invalid MsgType
12 = XML Validation error
13 = Tag appears more than once
14 = Tag specified out of required order
15 = Repeating group fields out of order
16 = Incorrect NumInGroup count for repeating group
17 = Non "data" value includes field delimiter (SOH character)
(Note other session-level rule violations may exist in which case SessionRejectReason is not specified)

Reject

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 3
45	RefSeqNum	Y	MsgSeqNum of rejected message
58	Text	N	Where possible, message to explain reason for rejection
	Standard Trailer	Y	

8.8 SEQUENCE RESET-GAPFILL

The sequence reset message is used by the sending application to reset the incoming sequence number on the opposing side. This message has two modes: "Sequence Reset-Gap Fill" when GapFillFlag is 'Y' and "Sequence Reset-Reset" when GapFillFlag is N or not present. The "Sequence Reset-Reset" mode should **ONLY** be used to recover from a disaster situation which cannot be otherwise recovered via "Gap Fill" mode. The sequence reset message can be used in the following situations:

- ✓ nDuring normal resend processing, the sending application may choose not to send a message (e.g. an aged order). The Sequence Reset - Gap Fill is used to mark the place of that message.
- ✓ During normal resend processing, a number of administrative messages are not resent, the Sequence Reset – Gap Fill message is used to fill the sequence gap created.

- ✓ In the event of an application failure, it may be necessary to force synchronization of sequence numbers on the sending and receiving sides via the use of Sequence Reset – Reset

Sequence Reset

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 4
123	GapFillFlag	N	
36	NewSeqNo	Y	
	Standard Trailer	Y	

8.9 LOGOUT MESSAGE

The logout message initiates or confirms the termination of a FIX session. Disconnection without the exchange of logout messages should be interpreted as an abnormal condition. Before actually closing the session, the logout initiator should wait for the opposite side to respond with a confirming logout message. This gives the remote end a chance to perform any Gap Fill operations that may be necessary. The session may be terminated if the remote side does not respond in an appropriate timeframe.

After sending the Logout message, the logout initiator should not send any messages unless requested to do so by the logout acceptor via a ResendRequest.

Logout

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 5
58	Text	N	
	Standard Trailer	Y	

9 TEST CASES

'est case	Mandatory / Optional	Condition/Stimulus	Expected Behavior
.eceive **lessage tandard leader**	Mandatory	a. *MsgSeqNum* received as expected	Accept *MsgSeqNum* for the message
		b. *MsgSeqNum* higher than expected	Respond with **Resend Request** message
		c. *MsgSeqNum* lower than expected without *PossDupFlag* set to Y	1. Disconnect without sending a message 2. Generate an "error" condition in test output.
		d. Garbled message received	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages. 2. Generate a "warning" condition in test output.
		e. *PossDupFlag* set to Y; *OrigSendingTime* specified is less than *SendingTime* Note: *OrigSendingTime* should be earlier than *SendingTime* unless the message is being resent within the same second during which it was sent.	1. Accept the message.
		f. *PossDupFlag* set to Y; *OrigSendingTime* specified is greater than *SendingTime* and *MsgSeqNum* lower than expected Note: *OrigSendingTime* should be earlier than *SendingTime* unless the message is being resent within the same second during which it was sent.	1. Send **Reject (session-level)** message referencing inaccurate *SendingTime* (> FIX 4.2: SessionRejectReason = "SendingTime accuracy problem") 2. Increment inbound *MsgSeqNum* 3. Send Logout message referencing inaccurate SendingTime value 4. Wait for Logout message response (note likely will have inaccurate SendingTime) or wait 2 seconds whichever comes first 5. Disconnect Generate an "error" condition in test output.

		g. *PossDupFlag* set to Y and *OrigSendingTime* not specified Note: Always set *OrigSendingTime* to the time when the message was originally sent-not the present *SendingTime* and set *PossDupFlag* = "Y" when responding to a **Resend Request**	1. Send **Reject (session-level)** message referencing missing *OrigSendingTime* (>= FIX 4.2: SessionRejectReason = "Required tag missing") 2. Increment inbound *MsgSeqNum*
		h. *BeginString* value received as expected and specified in testing profile and matches *BeginString* on outbound messages.	Accept *BeginString* for the message
		i. *BeginString* value (e.g. "FIX.4.2") received did not match value expected and specified in testing profile or does not match *BeginString* on outbound messages.	1. Send Logout message referencing incorrect BeginString value 2. Wait for Logout message response (note likely will have incorrect BeginString) or wait 2 seconds whichever comes first 3. Disconnect 4. Generate an "error" condition in test output.
		j. *SenderCompID* and *TargetCompID* values received as expected and specified in testing profile.	Accept *SenderCompID* and *TargetCompID* for the message
		k. *SenderCompID* and *TargetCompID* values received did not match values expected and specified in testing profile.	1. Send **Reject (session-level)** message referencing invalid *SenderCompID* or *TargetCompID* (>= FIX 4.2: SessionRejectReason = "CompID problem") 2. Increment inbound *MsgSeqNum* 3. Send Logout message referencing incorrect SenderCompID or TargetCompID value 4. Wait for Logout message response (note likely will have incorrect SenderCompID or TargetCompID) or wait 2 seconds whichever comes first 5. Disconnect 6. Generate an "error" condition in test output.

		l. *BodyLength* value received is correct.	Accept *BodyLength* for the message
		m. *BodyLength* value received is not correct.	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages 2. Generate a "warning" condition in test output.
		n. *SendingTime* value received is specified in UTC (Universal Time Coordinated also known as GMT) and is within 2 minutes of atomic clock-based time.	Accept *SendingTime* for the message
		o. *SendingTime* value received is either not specified in UTC (Universal Time Coordinated also known as GMT) or is not within 2 minutes of atomic clock-based time. Rational: Verify system clocks on both sides are in sync and that SendingTime must be current time	1. Send **Reject (session-level)** message referencing inaccurate *SendingTime* (> FIX 4.2: SessionRejectReason "SendingTime accuracy problem") 2. Increment inbound *MsgSeqNum* 3. Send Logout message referencing inaccurate SendingTime value 4. Wait for Logout message response (note likely will have inaccurate SendingTime) or wait 2 seconds whichever comes first 5. Disconnect 6. Generate an "error" condition in test output.
		p. *MsgType* value received is valid (defined in spec or classified as user-defined). q. *MsgType* value received is not valid (defined in spec or classified as user-defined).	Accept *MsgType* for the message 1. Send **Reject (session-level)** message referencing invalid *MsgType* (> FIX 4.2: SessionRejectReason "Invalid MsgType") 2. Increment inbound *MsgSeqNum* 3.Generate a "warning" condition in test output.

		r. *MsgType* value received is valid (defined in spec or classified as user-defined) but not supported or registered in testing profile.	*1)* If < FIX 4.2 *a)* Send **Reject (session-level)** message referencing valid but unsupported *MsgType* *2)* If >= FIX 4.2 *a)* Send **Business Message Reject** message referencing valid but unsupported *MsgType* (>= FIX 4.2: BusinessRejectReason "Unsupported Message Type") 3) Increment inbound *MsgSeqNum* 4) Generate a "warning" condition in test output.
		s. *BeginString, BodyLength,* and *MsgType* are first three fields of message.	Accept the message
		t. *BeginString, BodyLength,* and *MsgType* are not the first three fields of message.	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages 2. Generate a "warning" condition in test output.
:ive **sage** **ıdard** **ıler**	Mandatory	a. Valid *CheckSum*	Accept Message
		b. Invalid *CheckSum*	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages 2. Generate a "warning" condition in test output.
		c. Garbled message	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages 2. Generate a "warning" condition in test output.

		d. *CheckSum* is last field of message, value has length of 3, and is delimited by <SOH>.	Accept Message
		e. *CheckSum* is not the last field of message, value does not have length of 3, or is not delimited by <SOH>.	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum* and continue accepting messages 2. Generate a "warning" condition in test output.
	Mandatory	a. Valid **Logon** message	1. Respond with **Logon** response message 2. If *MsgSeqNum* is too high then send **Resend Request**
		b. Invalid **Logon** message	1. Generate an "error" condition in test output. 2. (Optional) Send **Reject** message with *RefMsgSeqNum* referencing **Logon** message's *MsgSeqNum* with *Text* referencing error condition 3. Send **Logout** message with *Text* referencing error condition 4. Disconnect
		c. First message received is not a Logon message.	1. Log an error "first message not a logon" 2. Disconnect
at	Mandatory	a. No data sent during preset heartbeat interval (*HeartBeatInt* field)	Send **Heartbeat** message
		Valid **Heartbeat** message	Accept **Heartbeat** message

;st	Mandatory	No data received during preset heartbeat interval (*HeartBeatInt* field) + "some reasonable period of time"	1. Send **Test Request** message 2. Track and verify that a **Heartbeat** with the same *TestReqID* is received (may not be the next message received)
d :st ge	Mandatory	Valid Resend Request	Respond with application level messages and **SequenceReset**-*Gap Fill* for admin messages in requested range according to "Message Recovery" rules.
nce (Gap	Mandatory	a. Receive **Sequence Reset (Gap Fill)** message with NewSeqNo > MsgSeqNum and MsgSeqNum > than expect sequence number	Issue **Resend Request** to fill gap between last expected MsgSeqNum & received MsgSeqNum.
		b. Receive **Sequence Reset (Gap Fill)** message with NewSeqNo > MsgSeqNum and MsgSeqNum = to expected sequence number	Set next expected sequence number = NewSeqNo
		c. Receive **Sequence Reset (Gap Fill)** message with NewSeqNo > MsgSeqNum and MsgSeqNum < than expected sequence number and PossDupFlag = "Y"	Ignore message
	Mandatory	Initiate Logout	1.Send **Logout** message 2. Disconnect

10 SYSTEM IMPLEMENTATION AND TESTING

10.1 SYSTEM IMPLEMENTATION

As a policy every product in the company ready for release undergoes a versioning and release management process. The product is versioned and then implemented in the client location. A complete set of operational documentation, user's manual and guidelines are supplied. Professionals exclusively give user training to a few in the client place from the company.

IMPLEMENTATION PROCEDURES

The project undergoes a versioning and release management before it is delivered to the client. It is a process of identifying and keeping track of different versions and releases of the system. And the released product usually includes configuration files defining how the release should be configured for particular installations. Data files needed for successful operations. An Installation program, which is used to help install the system on the target hardware. Electronic and paper documentation describing the system. All these information are made available on a medium, which can be read and understood by the customer for the software.

The following factors are considered before implementation. Checking if all the components which make up the system been included, if the appropriate version of each required component been included, are the data objects included, etc...An installation program is created and the entire kit is delivered to the client.

USER TRANING

The kit delivered consists of a complete guide on the new system developed. A through training on the new system is given to a representative from each of the user area and an overall demo given to the entire team. The queries from the audience were answered and hints given on various issues. Special training was given to the admin staff

that is to play the role of super user. The configuration details and trouble shooting methodologies were explained and his performance absorbed. The user manual was completely explained and doubts cleared for the same. Installing and uninstalling the package and taking a backup of the data were demonstrated to the super user. Various possible exceptions and the possible causes for it from the user's end were explained. The various user environments and the right of access specified to each user was clearly explained and demonstration given to the team on different user environments. Instructions on successful operation of the system and trouble shooting methodologies were thus discussed.

OPERATIONAL DOCUMENTATION

Properly produced and maintained system documentation is a tremendous aid to maintenance engineers. The system documentation includes all the documents describing the implementation of the system from the requirements specification to the final acceptance test plan.

A complete set of Operational Documentation was prepared for the client, which included the features of the system, the access rights allocated for various users and trouble shooting details. The special features of the system were highlighted. A step-by-step procedure was included in the documentation for FIX Message Creation, Validation and Monitoring. The documentation is prepared keeping in mind users who have little or no knowledge of computers.

The operational documentation includes a document describing the overall architecture, a maintenance guide, a user manual for operations like FIX Message Creation, Validation, Session and Exceptions and their causes and solution. The way the FINSwitch receives messages and processes them are explained with the help of Architecture and Run-Time views. A clear picture of the system and its functionalities are thus provided.

10.2 SYSTEM TESTING

TESTING PROCESS

Except for small software, systems should not be tested as a single, monolithic unit. Large systems are built out of sub-systems, which in turn are built out of sub systems, composed of procedures and functions. The testing process should therefore proceed in stages where testing is carried out incrementally in conjunction with the system implementation.

There are the five testing stages and defects are discovered at any stage, they require program modifications to correct them and this require other stages in the testing process to be repeated. The process therefore in an interactive one with information being fed back from later stages to earlier parts of the process.

The stages in the testing process are:

- ✓ Unit Testing
- ✓ Module Testing
- ✓ System Testing
- ✓ Acceptance Testing

UNIT TESTING

Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components. With respect to this project, the individual functions are treated as component and were tested.

MODULE TESTING

A module is a collection of dependent components such as an object class, an abstract data type or some looser collection of procedures and functions. A module encapsulates related components so it can be tested with other system components.

Each module in the FINSwitch works successfully and dependent components as object and class are checked.

SYSTEM TESTING

The sub-systems are integrated to make up the entire system. The testing process is concerned with finding errors, which result from unanticipated interactions between sub-system and system components. It is also concerned with validating that the system meets its functional and non-functional requirements. After integrating of the above sub-systems with the whole systems, the entire system is tested for error.

ACCEPTENCE TESTING

This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system procurer rather than simulated data. Acceptance testing may reveal errors and omissions in the system requirements definition because the real data exercises the system in different ways from testing the data. Acceptance testing may meet the users need or the system performance is unacceptable.

DEFECT TESTING

Defect testing is intended to exercise a system so that later defects are exposed before the system is delivered. These contrasts with validation testing which is intended to demonstrate that the system meets its requirement. Validation testing requires the system to perform correctly using given acceptance test cases. A successful defect test is a test, which causes a system to perform incorrectly and hence exposes the defects. It demonstrates the presence, not absence of program faults.

Various values, within the limit and exceeding the limit were provided repeatedly to individual components of data acquisition. These brought out the defects in the system and were corrected. Two approaches to defect testing are:

BLACK- BOX TESTING

It relies on the specification of the system or component, which being tested to derive test cases. The system is 'black-box' whose behavior can only be determined by studying its inputs and the related outputs. This is also called as functional testing because mathematical functional can be specified using only inputs and outputs.

Following black-box methods were applied to both the modules to test arrays:

- Usage of only one value of entire array. This proved that the program works for an exceptional array.
- Usage of different arrays of different sizes. This decreased the chances that the program with defect would accidentally produce a correct output because of some characteristic of the inputs.
- First, middle and last elements were accessed and any problems due to the boundary effects were delivered.

STRUCTURAL TESTING

This is the complementary approach to black box testing and is sometimes called structural, white-box or glass-box testing. The tester can analyze the code and the use knowledge about the structural of the component to derive test data.

The advantage of structural testing is that an analysis of the code can be used to find how many test cases are needed to guarantee a given level of test courage. A dynamic analysis can then used to measure the extend of this coverage and help with test case design.

PATH TESTING

Path testing is a white-box testing strategy whose objective is to exercise every independent execution path through the component. If every independent path is executed then all the statements in the program must have been executed at least one. Furthermore, all conditional statements are tested for both true and false causes. This helped to improve the program efficiency with respect to time complexity and memory usage.

CONCLUSION

FINSwitch has been successfully designed, implemented and tested. The software developed is found to work effectively and efficiently.

The FINSwitch library helps to create FIX Messages in a very effective way. The classes and methods provided by the library helps the end user to a great extent in developing and modifying the Messages. FINSwitch helps the market participants to easily go with their day-to day business activities. The electronic exchange of information in securities industries is greatly favored by this FIX Engine.

FINSwitch plays the role of a messaging, translation and routing intermediary between a financial institution's trading/back office applications that communicate through proprietary API and systems that use FIX, FIXML protocols. These FIX/FIXML systems could be applications within an organization (order routing gateways, crossing engines etc) and external destinations (broker-dealers, investment managers, post-trade intermediaries, execution destinations etc) working on FIX/FIXML.

FINSwitch supports the translation and routing of bi-directional message flows wherein messages emanating in application proprietary formats are converted to FIX/FIXML and messages based on FIX/FIXML are converted to the pertinent application API. The direction (routing) of message flows intelligently, for both inbound and outbound messages and between internal applications is a core capability of the product.

SCOPE FOR FUTURE DEVELOPMENT

The primary objective of FINSwitch is to facilitate the electronic exchange of information related to securities transactions. Since FINSwitch is a product it keeps on adding new features and functionality towards the end of each release.

The forthcoming releases for the product offer high protocol extensibility with support for SWIFT ISO 15022, FpML, OFX & IFX. Protocol adaptors for connectivity to the virtual matching Utilities (VMUs) - OMGEO and GSTP are also planned in the future releases.