EXHIBIT 30

UNITED STATES DISTRICT COURT
SOUTHERN DISTRICT OF NEW YORK

In re: **BERNARD L. MADOFF INVESTMENT SECURITIES LLC,** **Debtor,**	**Adv. Pro. No. 08-01789 (BRL)** **SIPA LIQUIDATION** **(Substantively Consolidated)** **Adv. Pro. No. 10-5287 (BRL)**
IRVING H. PICARD, Trustee for the Liquidation of Bernard L. Madoff Investment Securities LLC, **Plaintiff,** **v.** **SAUL B. KATZ, et al.,** **Defendants.**	**11-CV-03605 (JSR) (HBP)**

Bruce Dubinsky Deposition Binder

Tabs 81-137

Part III of III

ATTORNEY WORK PRODUCT
PRIVILEGED & CONFIDENTIAL

DUFF & PHELPS

EXHIBIT
D-25C
1-12-12

```
package MadoffFixMonitor;

import java.util.ArrayList;
import java.util.Random ;
import java.util.TreeMap;
import java.util.StringTokenizer;
import java.io.* ;
import java.util.LinkedList;

//import org.apache.jasper.Constants;


import com.cameronsystems.fix.message.Message ;
import com.cameronsystems.fix.oms.messagebrowser.MessagePanel;

import com.cameronsystems.fix.configuration.Constants;


public class MADOFFRandomSimulationUtility {
        /**
         * Constructs a activity element.
         *
         */

// Simulation

   public static ArrayList alAllEQTDList = new ArrayList() ;
   public static ArrayList alOneEQTDList = new ArrayList() ;

   public static int iLocAccountNum = -1 ;
   public static int iLocCusip = -1 ;
   public static int iLocTradeDate = -1 ;
   public static int iLocTradePrice = -1 ;
   public static int iLocTradeQty = -1 ;
   public static int iLocSecurityDesc = -1 ;
   public static int iLocSettleDate = -1 ;
   public static int iLocSide = -1 ;
   public static String sTRADE_CUSIP = "" ;
   public static String sTRADE_DATE = "" ;
   public static String sTRADE_PRICE = "" ;
   public static String sTRADE_QTY = "" ;
   public static String sSECURITY_DESC = "" ;
   public static String sSETTLE_DATE = "" ;
   public static String sTRADE_SIDE = "" ;
   public static double dTRADE_PRICE = 0.0 ;

   public static String sThisTradeCusip = null ;
   public static String sThisTradeDate = null ;
   public static String sThisTradePrice = null ;
   public static String sThisTradeQty = null ;
   public static String sThisSecurityDesc = null ;
   public static String sThisSettleDate = null ;
   public static String sThisSide = null ;
   public static double dTotalQty = 0 ;
   public static double dTotalPrice   = 0 ;
```

```
    public static double dThisTmpTotalQty = 0 ;

    public static double dThisTradeQty = 0 ;
    public static double dThisTradePrice   = 0 ;

    public static int iRoundedOrdersPerOddLot = 0 ;
    public static int iRoundedOrderCount = 0 ;

    public static String sTITLE_FIELDS = null ;

    public static Random generator = new Random() ;
    public static Random priceGenerator = new Random() ;
    public static Random intGenerator = new Random() ;
    public static Random idGenerator = new Random() ;
    public static Random timeGenerator = new Random() ;
    public static Random liveGenerator = new Random() ;
    public static String ACCOUNT_NUMBER = "account number" ;
    public static String QUANTITY = "quantity" ;
    public static String PRICE = "price" ;
    public static String CUSIP = "cusip" ;
    public static String TD_DATE =   "t/d" ;
    public static String SECURITY_DESC = "security description" ;
    public static String SEC_DESC = "sec desc" ;
    public static String SETTLE_DATE = "s/d" ;
    public static String SIDE =   "side" ;
    public static String sORDER_TITLE = "Number,T/D,Security
Description,Quantity,Price,S/D,Cusip,Entry Time,Live Period,Cancel Time\r\n" ;
    public static String sSTATS_TITLE = "Security Description,Cusip,Order Count,Total Trade Qty,Original
Trae Price,Trade QtyPrice,Original Order QtyPrice,Original Order QtyPrice Diff,Final Order QtyPrice,Final
QtyPrice Diff\r\n" ;
    public static BufferedWriter bwRandomOrderOutFile = null ;
    public static BufferedWriter bwReportOutFile = null ;
    public static BufferedWriter   bwRandomStatsFile = null ;
    public static TreeMap tmID = null ;
    public static MadoffDB madoffDB = new MadoffDB() ;
    public static boolean loadEQTDFile(String sEQ_TD) {
        boolean bSuccess = false ;
        lStartEntryTime = getLongTime(StartEntryTime) ;
        lEndEntryTime = getLongTime(EndEntryTime) ;
        lEntryPeriod = lEndEntryTime - lStartEntryTime ;
        lLivePeriodRange = getLivePeriodRange() ;
        msgFIFO.clear() ;
        alAllEQTDList.clear() ;
            try {
                    BufferedReader eqListFile = new BufferedReader(new FileReader(sEQ_TD));
                    while (eqListFile.ready()) {
                            String strKey = eqListFile.readLine().toString().trim();
                            if (!strKey.startsWith("#")   && !strKey.trim().equals("") ) {
                                  alAllEQTDList.add(strKey) ;
//                                msgFIFO.put(strKey) ;
                            } else if (strKey.toLowerCase().indexOf("price") != -1 &&
strKey.toLowerCase().indexOf("account number") != -1) {
                                  sTITLE_FIELDS = strKey ;
                                  parseTitleFieldsLoc() ;
//                                if (messageParser!=null && !messageParser.isAlive())
messageParser.start() ;
```

```
                        }
                    }
                    eqListFile.close() ;
                    tmID = new TreeMap() ;
            } catch (Exception e) {

            }
//          if (alAllEQTDList.size() > 0)   bSuccess = true ;
//          lStartEntryTime = getLongTime(StartEntryTime) ;
//          lEndEntryTime = getLongTime(EndEntryTime) ;
//          lEntryPeriod = lEndEntryTime - lStartEntryTime ;
//                  lLivePeriodRange = getLivePeriodRange() ;
            return bSuccess ;
    }

    public static boolean bRandomizePrice = true ;
    public static String sDBType = null ;
    public static void loadConfig(String sConfigFile) {
            try {
                    sDBType = null ;
                        BufferedReader configFile = new BufferedReader(new FileReader(sConfigFile));
                        while (configFile.ready()) {
                                String strKey = configFile.readLine().toString().trim();

                                try {

                                        if (!strKey.startsWith("#")   && !strKey.trim().equals("") ) {

                                                if (strKey.toLowerCase().indexOf("startentrytime")!= -1) {
StartEntryTime = strKey.substring(strKey.indexOf("=")+1).trim() ;
                                                } else if (strKey.toLowerCase().indexOf("endentrytime")!= -1){
EndEntryTime = strKey.substring(strKey.indexOf("=")+1).trim() ;
                                                } else if (strKey.toLowerCase().indexOf("maxqty")!= -1){ String
sMaxQty = strKey.substring(strKey.indexOf("=")+1).trim() ; iMaxQty = Integer.parseInt(sMaxQty) ;
                                                } else if (strKey.toLowerCase().indexOf("minqty")!= -1){ String
sMinQty = strKey.substring(strKey.indexOf("=")+1).trim() ; iMinQty = Integer.parseInt(sMinQty) ;
                                                } else if
(strKey.toLowerCase().indexOf("maxnumoforderperoddlot")!= -1){ String sMaxNumOfOrderPerOddLot =
strKey.substring(strKey.indexOf("=")+1).trim() ; iMaxNumOfOrderPerOddLot =
Integer.parseInt(sMaxNumOfOrderPerOddLot) ;
                                                } else if (strKey.toLowerCase().indexOf("randomizeprice")!= -1){
String sRandomizePrice = strKey.substring(strKey.indexOf("=")+1).trim() ; if
(sRandomizePrice.equalsIgnoreCase("false")) bRandomizePrice = false ;
                                                } else if (strKey.toLowerCase().indexOf("maxdigitnumber")!= -1){
String sMaxNumOfDigits = strKey.substring(strKey.indexOf("=")+1).trim() ; iMaxNumberDigit =
Integer.parseInt(sMaxNumOfDigits) ;
                                                } else if (strKey.toLowerCase().indexOf("livestartminute")!= -1){
String sLiveStartMinute = strKey.substring(strKey.indexOf("=")+1).trim() ; lLiveStartMinute =
Integer.parseInt(sLiveStartMinute) ;
                                                } else if (strKey.toLowerCase().indexOf("liveendminute")!= -1){
String sLiveEndMinute = strKey.substring(strKey.indexOf("=")+1).trim() ; lLiveEndMinute =
Integer.parseInt(sLiveEndMinute) ;
                                                }

                                        } else if (strKey != null && strKey.toLowerCase().indexOf("order") != -1
&& strKey.toLowerCase().indexOf("file") != -1) sDBType = "Order" ;
                                                    else if (strKey != null && strKey.toLowerCase().indexOf("opt") !=
-1 && strKey.toLowerCase().indexOf("file") != -1) sDBType = "Opt" ;
                                        } catch (Exception e) {
                                        }
                                }
                        configFile.close() ;
                } catch (Exception e) {

                }

        }

        public static class RandomGenerator extends Thread {
                public void run() {
                        this.setName("RandomGenerator") ;
                        try {
                                this.setPriority(1) ; // THREAD_PRIORITY_LOWEST
//                               while (true) {
                                        try {
                                                if (sRandomType != null &&
sRandomType.toLowerCase().indexOf("ord") != -1)
                                                {

                                                        madoffDB.startFIXDatabase("dbName.txt") ;
                                                        loadConfig("random_order_config.txt") ;
                                                        createNewOutputFile("RANDOM_ORDER.csv") ;
//                                                      parseTitleFieldsLoc() ;
                                                        loadEQTDFile("ORDER_EQ_TD.csv") ;
                                                        generateAllRandomOrderSequence() ;
                                                } else if (sRandomType != null &&
sRandomType.toLowerCase().indexOf("opt") != -1) {
                                                        madoffDB.startFIXDatabase("dbName.txt") ;
                                                        loadConfig("random_opt_config.txt") ;
                                                        createNewOutputFile("RANDOM_OPT.csv") ;
//                                                      parseTitleFieldsLoc() ;
                                                        loadEQTDFile("OPT_EQ_TD.csv") ;
                                                        generateAllRandomOrderSequence() ;

                                                }
                                        } catch (Exception e) {
                                        }

                                        Thread.yield() ;
//                               }
                        } catch (Exception e) {
                        }
                }
        }

        private static MessageParser messageParser = new MessageParser() ;

        private static Object inFIXObj = null ;
        public static class MessageParser extends Thread {
                public void run() {
                        this.setName("MessageParser") ;
```

```
				try {
					this.setPriority(1) ; // THREAD_PRIORITY_LOWEST
					 while (true) {
						try {
							if (madoffDB.getFIFOSize() < 5000) {

								if ((inFIXObj=msgFIFO.get())!=null ) {
									generateRandomSequence((String )inFIXObj) ;
								 }
//									 addFIXMsgIntoDB(inFIXObj) ;
							}
						} catch (Exception e) {
						}

						Thread.yield() ;
					}
				} catch (Exception e) {
				}
			}
	}
//
//	private String sFieldName = "insert into dbo.FIX_MESSAGE
(MadoffTime,SendingTime,Direction,InterfaceNumber,Version,MsgSeqNum,SenderCompID,SenderSubI
D,TargetCompID,TargetSubID,PossDupe,PossResend,OnBehalfCompID,OnBehalfSubID,MsgType,OrdT
ype,Side,Qty,LastQty,Symbol,SymbolSfx,ClOrdID,OriClOrdID,ClientID,Price,LastPx,StopPx,AvgPx,LastM
kt,ExecBroker,TransactTime,TimeInForce,HandlInst,ExecInst,Rule80A,SecurityIDSource,SecurityID,Effec
tiveTime,ExpireTime,OrigTime,OrderID,ExecID,ExecType,OrdStatus,ExecTransType,LeavesQty,CumQty
,ClearingFirm,OriginalMsg,UNIQUE_ID) values " ;
//	private String sOrderName = "insert into dbo.Orders
(Number,TradeDate,SecurityDescription,Quantity,Price,SettleDate,Cusip,EntryTime,LivePeriod,CancelTi
me,MadoffTime,UNIQUE_ID) values " ;
//	private String sOptName = "insert into dbo.Options
(Number,TradeDate,SecurityDescription,Quantity,Price,SettleDate,Cusip,EntryTime,LivePeriod,CancelTi
me,MadoffTime,UNIQUE_ID) values " ;
//	private String sFieldValue = null ;
//	String sSQLinsert = null ;
//
//	private void addFIXMsgIntoDB(Object dbMsg) {
//		try {
//			sFieldValue = (String) dbMsg ;
//			if (sFieldValue.startsWith("Order")) {
//				if (iOrderUniqueID >= 0) {
//					sFieldValue = sFieldValue.replaceAll("Orders,","") ;
//
//					if (sFieldValue != null) {
//						sSQLinsert = sOrderName+sFieldValue ;
//						sFail = executeUpdateNoThrow(sSQLinsert);
//						if (sFail!=null) {
//						}
//					}
//				}
//			} else if (sFieldValue.startsWith("Opt")) {
//				if (iOptUniqueID >= 0) {
//					sFieldValue = sFieldValue.replaceAll("Opt,","") ;
//
//					if (sFieldValue != null) {
```

```
//                                          sSQLinsert = sOptName+sFieldValue ;
//                                          sFail = executeUpdateNoThrow(sSQLinsert);
//                                          if (sFail!=null) {
//                                          }
//                                   }
//                            }
//                    }
//          } catch (Exception e) {
//          }
//    }
//
//

          private static NoticeFIFO msgFIFO = new NoticeFIFO() ;

     private static class NoticeFIFO {
                    private LinkedList Buffer;

                    public NoticeFIFO() {
                      Buffer = new LinkedList() ;
                    }

                    public NoticeFIFO(int size) {
                            Buffer = new LinkedList() ;
                    }

                    public synchronized void put(Object value) {
                             try {
                                  Buffer.addLast(value) ;
                                  notify() ;
                             } catch (Exception e) {
                             }
                    }

                   public synchronized void clear() {
                            try {
                             Buffer.clear() ;
                             notify() ;
                            } catch (Exception e) {
                            }
                   }

                    public synchronized Object get() {
                             Object RetVal = null ;
                                  try {
                                     if (Buffer.isEmpty())    {wait();}
                                     RetVal = Buffer.removeFirst();
                                  } catch (Exception e) {
                                  }
                      return (RetVal);
                    }
          }
//
//    private NoticeFIFO fixDBMsgFIFO = new NoticeFIFO() ;
//
//
```

```
//   private NoticeFIFO fixMsgFIFO = new NoticeFIFO() ;
//   private FIXMessageDB fixMessageDB = new FIXMessageDB() ;
//
//   public boolean getFIXDB() {
//        return bFIXDB ;
//   }
//
//   public void setFIXDB(boolean bStatus) {
//        bFIXDB = bStatus ;
//   }
//   private static boolean bFIXDB = false ;
//
//   public void startFIXDatabase(String sDBNameFile) {
//        try {
//                  File fixDatabaseFile = new File(sDBNameFile) ;
//                  BufferedReader brDB =new BufferedReader(new FileReader(sDBNameFile));
//                  String sDB = null ;
//                  if (fixDatabaseFile.exists() ) {
//                            sDB = brDB.readLine() ;
//                            if (sDB != null && !sDB.trim().equals("")) {
//                                      bFIXDB = true ;
//                                      connectFIXDb(sDB) ;
//                                      if (fixMessageDB!=null && !fixMessageDB.isAlive())
//                                                fixMessageDB.start() ;
//                            }
//                  } else {
//                  }
//        } catch (Exception e) {
//        }
//   }
//
//   private Connection dbConn ;
//   private Statement dbStatement ;
//   private static int iFIXUniqueID = -1 ;
//   private static int iOrderUniqueID = -1 ;
//   private static int iOptUniqueID = -1 ;
//   private String sFail = null ;
//
//   public boolean connectFIXDb(String sDB) throws SQLException, ClassNotFoundException { //[Test
Database Connection]
//        boolean bSuccess = true ;
//        String connString = null ;
//        String userName = null ;
//        String password = null ;
//        String dbDriver = null ;
//        try {
//
//                  String url = "jdbc:microsoft:sqlserver://";// SecurityMaster.getProperty(DB_URL,
"jdbc:microsoft:sqlserver://");
//                  userName = "smloader" ; // SecurityMaster.getProperty(DB_USER_NAME);
//                  String dbName = "SecurityMaster";   //SecurityMaster.getProperty(DB_INSTANCE);
//                  String host =   "AUDIT"; // SecurityMaster.getProperty(DB_HOST);
//                  if (sDB != null && !(sDB.trim().equals(""))) host = sDB ;
//                  String ipPort = "1433"; //SecurityMaster.getProperty(DB_IP_PORT);
//                  password = "sec@load"; // SecurityMaster.getProperty(DB_PASSWORD);
//                  dbDriver = "com.microsoft.jdbc.sqlserver.SQLServerDriver"; //
```

```
SecurityMaster.getProperty(DB_DRIVER, "com.microsoft.jdbc.sqlserver.SQLServerDriver");
//
//                      connString = url + host + ":" + ipPort + ";databaseName=" + dbName + ";";
//
//                      Class.forName(dbDriver);
//                      dbConn = DriverManager.getConnection(connString, userName, password);
//                      dbStatement = dbConn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
//          } catch (Exception e) {
//          }
//
//          try {
//                      iOrderUniqueID = lookupIfTablePopulated("Orders") ;
//                      iOptUniqueID = lookupIfTablePopulated("Options") ;
//                      sFail = null ;
//                      if (iOrderUniqueID <0) {
//                                  deleteDBTable ("Orders");
//                                  if (sFail != null) {
//                                              }
//                                  else {
//                                              disconnectDb() ;
//                                              Class.forName(dbDriver);
//                                              dbConn = DriverManager.getConnection(connString, userName,
password);
//                                              dbStatement =
dbConn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
//                                              iFIXUniqueID++ ;
//                                              }
//                                  }
//                      if (iOptUniqueID <0) {
//                                  deleteDBTable ("Options");
//                                  if (sFail != null) {
//                                              }
//                                  else {
//                                              disconnectDb() ;
//                                              Class.forName(dbDriver);
//                                              dbConn = DriverManager.getConnection(connString, userName,
password);
//                                              dbStatement =
dbConn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
//                                              iFIXUniqueID++ ;
//                                              }
//                                  }
//                      if (iOrderUniqueID > 0 && iOptUniqueID > 0) {
//                                  if (!isFIXTodayDate("Orders")) {
//                                              backupFIXDB() ;
//                                              deleteDBTable ("Orders");
//                                              iOrderUniqueID = lookupIfTablePopulated("Orders") ;
//                                  }
//                                  if (!isFIXTodayDate("Options")) {
//                                              backupFIXDB() ;
//                                              deleteDBTable ("Options");
//                                              iOptUniqueID = lookupIfTablePopulated("Options") ;
//                                  }
//
////                                iFIXUniqueID = lookupIfTablePopulated("FIX_MESSAGE") ;
////                          iFIXUniqueID++ ;
//              }
//          } catch (Exception e) {
//          }
//          return bSuccess ;
//  }
//
//  public void backupFIXDB() {
//          try {
//          } catch (Exception e) {
//          }
//  }
//
//
//  private boolean disconnectDb() {
//          boolean bSuccess = true ;
//          try {
//                  dbStatement.close() ;
//                  dbConn.close();
//          } catch (SQLException e) {
//            bSuccess = false ;
//          }
//          return bSuccess ;
//  }
//
//  public String executeUpdateNoThrow(String s) {
//          String sFail = null ;
//          try {
//            int i = dbStatement.executeUpdate(s);
//          } catch (SQLException e) {
//            sFail = "executeUpdateNoThrow :"+e+", "+s ;
//          }
//          return sFail ;
//  }
//
//  private int lookupIfTablePopulated(String sTable) {
//          int iReturn = -1 ;
//          try {
//            String select = "select count(*) from ["+sTable+"]" ;
//            ResultSet rs = null ;
//            try {
//                  rs = dbStatement.executeQuery(select);
//                  rs.next();
//                  iReturn = rs.getInt(1);
//                  rs.close();
//            } catch (Exception e) {
//            }
//
//          } catch (Exception e) {
//          }
//          return iReturn;
//  }
//
//  private final SimpleDateFormat timeFormatTodayDate_ = new   SimpleDateFormat("MM/dd/yy");
//
//  public boolean isFIXTodayDate(String sTable) // Get Database Date
//   {
//         boolean bToday = false ;
//         int iReturn = -1 ;
//         try {
//            String sTodayDate = timeFormatTodayDate_.format(new Date()) ;
//            String select = "select count(*) from dbo."+sTable+" where MadoffTime LIKE '%" +
sTodayDate+"%'" ;
//            ResultSet rs = null ;
//            try {
//                  rs = dbStatement.executeQuery(select);
//                  rs.next();
//                  iReturn = rs.getInt(1);
//                  rs.close();
//                  if (iReturn > 0) bToday = true ;
//            } catch (Exception e) {
//            }
//         } catch (Exception e) {
//         }
//         return bToday;
//
//   }
//
//   public void deleteDBTable (String sTable) {
//         try {
//                  sFail = executeUpdateNoThrow("delete from [securitymaster].[dbo].["+sTable+"]");
//                  if (sFail != null) {
//                        }
//                  else {
//                  }
//         } catch (Exception e) {
//         }
//   }

    public static void createNewOutputFile(String sOutputFileName) {
          try {
                    bwRandomOrderOutFile = new BufferedWriter( new FileWriter(new
File(sOutputFileName)));
                    bwRandomOrderOutFile.write(sORDER_TITLE+"\r\n" ) ;
                    bwRandomOrderOutFile.flush() ;


                    bwRandomStatsFile = new BufferedWriter( new FileWriter(new
File("RANDOM_STATS.csv")));
                    bwRandomStatsFile.write(sSTATS_TITLE+"\r\n" ) ;
                    bwRandomStatsFile.flush() ;

          } catch (Exception e) {
          }
    }

          public static boolean parseTitleFieldsLoc() {
                    boolean bSuccess = false ;
                    iLocAccountNum = -1 ;
                    iLocTradeQty = -1 ;
                    iLocTradePrice = -1 ;
                    iLocCusip = -1 ;
```

```
                iLocTradeDate = -1 ;
                iLocSecurityDesc = -1 ;
                iLocSettleDate = -1 ;
                iLocSide = -1 ;
                try {
                        StringTokenizer strToken = null;
                        String strOneElement = null ;
                        strToken = new StringTokenizer((String) sTITLE_FIELDS, ",\t");
                        int iLoc = 0 ;
                        while (strToken.hasMoreTokens()) {
                                strOneElement = strToken.nextToken() ;
                                if (strOneElement.toLowerCase().indexOf(ACCOUNT_NUMBER) != -1)
iLocAccountNum = iLoc ;
                                else if (strOneElement.toLowerCase().indexOf(QUANTITY) != -1)
iLocTradeQty = iLoc ;
                                else if (strOneElement.toLowerCase().indexOf(PRICE) != -1)
iLocTradePrice = iLoc ;
                                else if (strOneElement.toLowerCase().indexOf(CUSIP) != -1) iLocCusip =
iLoc ;
                                else if (strOneElement.toLowerCase().indexOf(TD_DATE) != -1)
iLocTradeDate = iLoc ;
                                else if (strOneElement.toLowerCase().indexOf(SECURITY_DESC) != -1
|| strOneElement.toLowerCase().indexOf(SEC_DESC) != -1) iLocSecurityDesc = iLoc ;
                                else if (strOneElement.toLowerCase().indexOf(SETTLE_DATE) != -1)
iLocSettleDate = iLoc ;
                                else if (strOneElement.toLowerCase().indexOf(SIDE) != -1) iLocSide =
iLoc ;
                                iLoc++ ;
                        }

                } catch (Exception e) {
                }

                if (iLocAccountNum != -1 && iLocTradeQty != -1 && iLocTradePrice != -1 && iLocCusip !=
-1 && iLocTradeDate != -1 && iLocSecurityDesc != -1 && iLocSettleDate != -1 ) bSuccess = true ;
                return bSuccess ;
        }

        public static boolean parseOneLine(String sOneLine) {
                boolean bSuccess = false ;
                sThisTradeQty = null ;
                sThisTradePrice = null ;
                sThisTradeCusip = null ;
                sThisTradeDate = null ;
                sThisSecurityDesc = null ;
                sThisSettleDate = null ;
                sThisSide = null ;
                try {
                        StringTokenizer strToken = null;
                        String strOneElement = null ;
                        strToken = new StringTokenizer((String) sOneLine, ",\t");
                        int iLoc = 0 ;
                        while (strToken.hasMoreTokens()) {
                                strOneElement = strToken.nextToken() ;
                                if (iLocTradeQty == iLoc) sThisTradeQty = strOneElement ;
                                else if (iLocTradePrice == iLoc) sThisTradePrice = strOneElement ;
```

```
                    else if (iLocCusip == iLoc) sThisTradeCusip = strOneElement ;
                    else if (iLocTradeDate == iLoc) sThisTradeDate = strOneElement ;
                    else if (iLocSecurityDesc == iLoc) sThisSecurityDesc = strOneElement ;
                    else if (iLocSettleDate == iLoc) sThisSettleDate = strOneElement ;
                    else if (iLocSide == iLoc) sThisSide = strOneElement ;
                    iLoc++ ;
                }
            } catch (Exception e) {
            }

            if (sThisTradeQty != null && sThisTradePrice != null && sThisTradeCusip != null &&
sThisSide != null&& sThisTradeDate != null && sThisSecurityDesc != null && sThisSettleDate != null)
bSuccess = true ;

            return bSuccess ;
        }

        public static void assignSignatureForNewRandomOrderSeq(String sThisTradeCusip, String
sThisTradeDate, String sThisTradePrice, String sThisSide) {
            try {

                sTRADE_CUSIP = sThisTradeCusip ;
                sTRADE_DATE = sThisTradeDate ;
                sTRADE_PRICE = sThisTradePrice ;
                sSECURITY_DESC = sThisSecurityDesc ;
                sSETTLE_DATE = sThisSettleDate ;
                sTRADE_SIDE = sThisSide ;
                dTRADE_PRICE = Double.parseDouble(sTRADE_PRICE) ;
            } catch (Exception e) {
            }
        }


        public static void resetTotalQty() {
            dTotalQty = 0 ;
        }

        public static void resetTotalPrice() {
            dTotalPrice = 0 ;
        }

        public static void addThisQtyPriceToTotal(String sThisTradeQty, String sThisTradePrice) {
            try {
                dThisTradePrice = Double.parseDouble(sThisTradePrice) ;
                dThisTradeQty = Double.parseDouble(sThisTradeQty) ;

                dCurrentSumTradeQtyPrice = dCurrentSumTradeQtyPrice +
dThisTradePrice*dThisTradeQty ;

                dTotalPrice = dTotalPrice+dThisTradePrice ;
                dTotalQty = dTotalQty+dThisTradeQty ;
            } catch (Exception e) {
            }

        }
```

```
	public static int iMaxQty = 12500 ;
	public static int iMinQty = 2500 ;
	public static long getOriginalRandomQty() {
		long lQty = Math.abs(generator.nextLong()) % iMaxQty ;
		if (lQty < iMinQty) lQty = iMinQty+lQty ;

		return lQty ;
	}

	public static long getRoundedQty(long lNewQty) {
		long lQty = 0 ;

		lQty = Math.round(lNewQty/100) * 100;

		return lQty ;
	}

	public static int iMaxNumOfOrderPerOddLot = 11 ;
	public static long generateOneRandomQty() {
		long lOneQty = 0 ;
		long lNewQty = getOriginalRandomQty() ;

		if (iRoundedOrderCount < iRoundedOrdersPerOddLot) {
			lOneQty = getRoundedQty(lNewQty) ;
			iRoundedOrderCount++ ;
		} else {
			iRoundedOrdersPerOddLot = Math.abs( intGenerator.nextInt()) %
iMaxNumOfOrderPerOddLot ;
			iRoundedOrderCount = 0 ;
			lOneQty = lNewQty ;
		}

		return lOneQty ;
	}

	public static int iDirectionMidIni = 11 ;
	public static int iDirectionMid = 11 ;
	public static int iDirectionMax = 23 ;
	public static int iAvgOrderNum = 0 ;
	public static int iCurrentOrderNum = 0 ;
	public static double getRandomPriceChange() {
		double dChange = Math.abs( priceGenerator.nextInt()) % 5 ;

		int iChangeDirection = Math.abs( intGenerator.nextInt()) % iDirectionMax ;
		if (iChangeDirection < iDirectionMid ) dChange = dChange*(-1) ;

		dChange = dChange/100.0 ;

		return dChange ;
	}

	public static double generateOneRandomPrice() {
		double dOnePrice = 0 ;

		double dPriceChange = getRandomPriceChange() ;
```

```
            dOnePrice = dTRADE_PRICE+dPriceChange ;
            return dOnePrice ;
    }

    public static double dTotalSumQtyPrice = 0.0 ;
    public static double dCurrentSumTradeQtyPrice = 0.0 ;
    public static double dCurrentSumOrderQtyPrice = 0.0 ;
    public static void resetParameters() {
            iDirectionMid = iDirectionMidIni ;
            dCurrentSumOrderQtyPrice = 0.0 ;
            dCurrentSumTradeQtyPrice = 0.0 ;
    }


    public static String StartEntryTime = "3:00:00" ;
    public static long lStartEntryTime = 0 ;
    public static long lEndEntryTime = 0 ;
    public static long lEntryPeriod = 0 ;
    public static String EndEntryTime = "8:00:00" ;
    public static String sEntryTime = "" ;
    public static String sLivePeriod = "" ;
    public static String sCancelTime = "" ;
    public static long lEntryTime = 0 ;
    public static long lLivePeriod = 0 ;
    public static long lLivePeriodRange = 0 ;
    public static long lCancelTime = 0 ;
    public static long lLiveStartMinute = 1 ; // minute
    public static long lLiveEndMinute = 15 ; // minutes
    public static long lLiveStartSec = 1 ; // minute
    public static long lLiveEndSec = 15 ; // minutes

    public static long getLivePeriodRange() {
            long lPeriod = 0 ;
            try {
                    lLiveStartSec = lLiveStartMinute*60 ;
                    lLiveEndSec = lLiveEndMinute*60 ;
                    lPeriod = lLiveEndSec - lLiveStartSec ;
            } catch (Exception e) {
            }
            return lPeriod ;
    }

    public static long getLongTime (String strTime) {
            long lTime = 0 ;
              try {
                    if (strTime != null && !strTime.equals("null") && !strTime.equals("")) {

                            String sHour = null;
                            String sMin = null;
                            String sSec = null;
//                          String sMilli = null;
                            StringTokenizer strToken = null;
                            try {
                                    strToken = new StringTokenizer((String) strTime, ":.,");
                            } catch (Exception e) {
                            }
```

```
                                        try {
                                                sHour = strToken.nextToken();
                                                sMin = strToken.nextToken();
                                                sSec = strToken.nextToken();
//                                              sMilli = strToken.nextToken();
                                        } catch (Exception e) {
                                        }
//                                      if (sMilli == null ) sMilli = "0" ;
                                        if (sSec == null) sSec = "0" ;
                                        if (sMin == null) sMin = "0" ;
                                        if (sHour == null ) sHour = "0" ;
                                        lTime =
Long.parseLong(sSec)+Long.parseLong(sMin)*60+Long.parseLong(sHour)*60*60;
                                }
                        } catch (Exception e) {
                        }
                        return lTime ;
                }

        public static String sHour = "" ;
        public static String sMin = "" ;
        public static String sSec = "" ;
        public static String getStringTimeFromSeconds(long lTime) {
                String sTime = "" ;
                try {
                        long lSec = lTime % 60 ;
                        long lTotalMin = lTime / 60 ;
                        long lMin = lTotalMin % 60 ;
                        long lHour = lTotalMin / 60 ;

                        sSec = String.valueOf(lSec) ;
                        if (sSec.length() == 1) sSec= "0"+sSec ;
                        sMin = String.valueOf(lMin) ;
                        if (sMin.length() == 1) sMin="0"+sMin ;

                        sTime = String.valueOf(lHour)+":"+sMin+":"+sSec ;
                } catch (Exception e) {

                }
                return sTime ;
        }

        public static void generateRandomTime() {
                try {
                        lEntryTime = lStartEntryTime+Math.abs(timeGenerator.nextLong()) %
lEntryPeriod ;
                        lLivePeriod = lLiveStartSec+Math.abs(liveGenerator.nextInt()) %
lLivePeriodRange ;
                        lCancelTime = lEntryTime + lLivePeriod ;
                        sEntryTime = getStringTimeFromSeconds(lEntryTime) ;
                        sLivePeriod = getStringTimeFromSeconds(lLivePeriod) ;
                        sCancelTime = getStringTimeFromSeconds(lCancelTime) ;
                } catch (Exception e) {
                }
        }
        public static String sNewOrder = null ;
        public static String sID = null ;
        public static void outputOneRandomOrder(long lThisTradeQty, double dThisTradePrice) {
                try {
                        sID = getNewID() ;
                        generateRandomTime() ;
                        sNewOrder =
sID+","+sTRADE_DATE+","+sSECURITY_DESC+","+lThisTradeQty+","+dThisTradePrice+","+sSETTLE_
DATE+","+sTRADE_CUSIP+","+sEntryTime+","+sLivePeriod+","+sCancelTime+ "\r\n" ;
                        bwRandomOrderOutFile.write(sNewOrder) ;   bwRandomOrderOutFile.flush() ;
                        writeDB(lThisTradeQty) ;
                } catch (Exception e) {
                }
        }

        public static String sDBValue = "" ;
        public static void writeDB(long lThisTradeQty) {
                try {
                        if (sDBType != null) {
                                sDBValue =
sDBType+"','"+sID+"','"+sTRADE_DATE+"','"+sSECURITY_DESC+"','"+lThisTradeQty+"','"+dThisTradePri
ce+"','"+sSETTLE_DATE+"','"+sTRADE_CUSIP+"','"+sEntryTime+"','"+sLivePeriod+"','"+sCancelTime+ "'"
;
                                madoffDB.addDB(sDBValue) ;
                        }
                } catch (Exception e) {
                }
        }

        public static void adjustParameters(long lThisTradeQty, double dThisTradePrice) {

                dCurrentSumOrderQtyPrice = dCurrentSumOrderQtyPrice +
lThisTradeQty*dThisTradePrice ;

                double dEstimatedRestOrderQtyPrice = (dTotalQty-dThisTmpTotalQty)*dTRADE_PRICE
;
                double dDiffCurrentOrderQtyPriceToOriginalTradeQtyPrice =
dCurrentSumOrderQtyPrice+ dEstimatedRestOrderQtyPrice - dTotalSumQtyPrice ;
                if (dDiffCurrentOrderQtyPriceToOriginalTradeQtyPrice > 50) {
                        iDirectionMid = iDirectionMax ;
                } else if (dDiffCurrentOrderQtyPriceToOriginalTradeQtyPrice < -50) {
                        iDirectionMid = 0 ;
                }
                else if (dDiffCurrentOrderQtyPriceToOriginalTradeQtyPrice > 10) {
                        iDirectionMid = iDirectionMid + 3 ;
                        if (iDirectionMid>iDirectionMax)
                                iDirectionMid = iDirectionMax ;
                        }
                else if (dDiffCurrentOrderQtyPriceToOriginalTradeQtyPrice < -10) {
                        iDirectionMid = iDirectionMid - 3 ;
                        if (iDirectionMid < 0)
                         iDirectionMid = 0 ;
                        } else
                        iDirectionMid = iDirectionMidIni ;
                }
        public static void outputRandomStats(double dOriginalCurrentQtyPrice, double dOriginalDiff) {
                try {
                        String sStats =
sSECURITY_DESC+","+sTRADE_CUSIP+","+iOrderCount+","+dTotalQty+","+dTRADE_PRICE+","+dTot
alSumQtyPrice+","+dOriginalCurrentQtyPrice+","
+dOriginalDiff+","+dCurrentSumOrderQtyPrice+","+dDiff+"\r\n";
                        bwRandomStatsFile.write(sStats) ;
                        bwRandomStatsFile.flush() ;
                } catch (Exception e) {
                }
        }

        public static double dPRICE_BAND = 0.05 ;
        public static void lastOrderProcessing() {
                try{
                        dThisTradeQty = dTotalQty - dThisTmpTotalQty ;
                        double dOriginalCurrentQtyPrice = dCurrentSumOrderQtyPrice +
dThisTradeQty*dThisTradePrice ;
                        double dOriginalDiff = dTotalSumQtyPrice - dOriginalCurrentQtyPrice ;
                        double dPriceAdjust = 0.0 ;
                        double dNewPrice = 0.0 ;
                        if (dOriginalDiff < 0) {
                                dPriceAdjust = (-1)*dOriginalDiff/dThisTradeQty ;
                                dNewPrice = dThisTradePrice - dPriceAdjust ;
                                if ((dTRADE_PRICE-dNewPrice) > dPRICE_BAND) dThisTradePrice =
dTRADE_PRICE -   dPRICE_BAND ;
                                else dThisTradePrice = dNewPrice ;
                        } else if (dOriginalDiff > 0) {
                                dPriceAdjust = dOriginalDiff/dThisTradeQty ;
                                dNewPrice = dThisTradePrice + dPriceAdjust ;
                                if ((dNewPrice - dTRADE_PRICE) > dPRICE_BAND) dThisTradePrice =
dTRADE_PRICE + dPRICE_BAND ;
                                else dThisTradePrice = dNewPrice ;

                        }

                        dCurrentSumOrderQtyPrice = dCurrentSumOrderQtyPrice +
dThisTradeQty*dThisTradePrice ;
                        dDiff = dTotalSumQtyPrice - dCurrentSumOrderQtyPrice ;
                        outputRandomStats(dOriginalCurrentQtyPrice, dOriginalDiff) ;

                } catch (Exception e) {
                }
        }

        public static double dDiff = 0.0 ;
        public static int iOrderCount = 0 ;
        public static void generateOneRandomOrderSequence() {
                try {
                        int iSign = 1 ;
                        if (dTotalQty < 0) {
                                dTotalQty = Math.abs(dTotalQty) ;
                                iSign = -1 ;
                        }
                        iOrderCount = 0 ;
```

```
                    dTotalSumQtyPrice = dTRADE_PRICE*dTotalQty ;
                    boolean bContinue = true ;
                    while (bContinue && dTotalQty > 0 ) {
                            iOrderCount++ ;
                            dThisTradeQty = generateOneRandomQty() ;
                            if (bRandomizePrice) dThisTradePrice = generateOneRandomPrice() ;
                            else dThisTradePrice = dTRADE_PRICE ;

                            if ((dThisTmpTotalQty+dThisTradeQty) > dTotalQty) {
                                    lastOrderProcessing() ;
                                    bContinue = false ;
                                    }
                            else {
                                    dThisTmpTotalQty = dThisTmpTotalQty + dThisTradeQty ;
                                    adjustParameters((long)dThisTradeQty, dThisTradePrice) ;
                            }
                            outputOneRandomOrder((long)(dThisTradeQty*iSign), dThisTradePrice)
;
                    }
            } catch (Exception e) {
            }
    }

    public static void prepareNewTradeSeq() {
            try{
                    dThisTmpTotalQty = 0 ;
                    alOneEQTDList.clear() ;
                    assignSignatureForNewRandomOrderSeq(sThisTradeCusip, sThisTradeDate,
sThisTradePrice, sThisSide) ;
                    resetTotalQty() ;
                    resetTotalPrice() ;
                    resetParameters() ;
                    dTotalQty = 0.0 ;
            } catch (Exception e) {
            }
    }


    public static void generateRandomSequence(String sOneLine) {
            try{
//                  sOneLine = (String) alAllEQTDList.get(iCount) ;
                    if ((dThisTradeQty>=0 && dTotalQty >= 0 || dThisTradeQty<=0 && dTotalQty <=
0) && parseOneLine(sOneLine) && sThisTradeCusip.equalsIgnoreCase(sTRADE_CUSIP) &&
sThisSide.equalsIgnoreCase(sTRADE_SIDE) && sThisTradeDate.equalsIgnoreCase(sTRADE_DATE)
&& sThisTradePrice.equalsIgnoreCase(sTRADE_PRICE)) {
                            alOneEQTDList.add(sOneLine) ;
                            addThisQtyPriceToTotal(sThisTradeQty, sThisTradePrice) ;
                    } else if (sThisTradeCusip != null &&   sThisSide != null &&   sThisTradeDate !=
null && sThisTradePrice != null && sThisTradeQty != null) {
                            generateOneRandomOrderSequence() ;
                            prepareNewTradeSeq() ;
                            addThisQtyPriceToTotal(sThisTradeQty, sThisTradePrice) ;
                            alOneEQTDList.add(sOneLine) ;
                    }

            } catch (Exception e) {
                }
        }

        public static void generateLastRandomSequence() {
                try {
                        if (sThisTradeCusip != null &&  sThisSide != null &&  sThisTradeDate != null &&
sThisTradePrice != null && sThisTradeQty != null) {
                                generateOneRandomOrderSequence() ;
                                prepareNewTradeSeq() ;
                        }

                        bwRandomOrderOutFile.close() ;
                        bwRandomStatsFile.close() ;

                } catch (Exception e) {
                }
        }

        public static void generateAllRandomOrderSequence() {
                try {
                        int iSize = alAllEQTDList.size() ;
                        int iCount = 0 ;
                        String sOneLine = null ;
                        while (iCount < iSize) {
                                sOneLine = (String) alAllEQTDList.get(iCount) ;
                                if ((dThisTradeQty>=0 && dTotalQty >= 0 || dThisTradeQty<=0 &&
dTotalQty <= 0) && parseOneLine(sOneLine) && sThisTradeCusip.equalsIgnoreCase(sTRADE_CUSIP)
&& sThisSide.equalsIgnoreCase(sTRADE_SIDE) &&
sThisTradeDate.equalsIgnoreCase(sTRADE_DATE) &&
sThisTradePrice.equalsIgnoreCase(sTRADE_PRICE)) {
                                        alOneEQTDList.add(sOneLine) ;
                                        addThisQtyPriceToTotal(sThisTradeQty, sThisTradePrice) ;
                                } else if (sThisTradeCusip != null &&  sThisSide != null &&
sThisTradeDate != null && sThisTradePrice != null && sThisTradeQty != null) {
                                        generateOneRandomOrderSequence() ;
                                        prepareNewTradeSeq() ;
                                        addThisQtyPriceToTotal(sThisTradeQty, sThisTradePrice) ;
                                        alOneEQTDList.add(sOneLine) ;
                                }
                                iCount++ ;
                        }

                        if (iCount==iSize && sThisTradeCusip != null &&  sThisSide != null &&
sThisTradeDate != null && sThisTradePrice != null && sThisTradeQty != null) {
                                generateOneRandomOrderSequence() ;
                                prepareNewTradeSeq() ;
                        }

                        bwRandomOrderOutFile.close() ;
                        bwRandomStatsFile.close() ;
                } catch (Exception e) {
                }

        }

        public static void loadReportInputFileNames() {
                try {
                        BufferedReader configFile = new BufferedReader(new
FileReader("report_input_files.txt"));
                        while (configFile.ready()) {
                                String strKey = configFile.readLine().toString().trim();

                                try {

                                        if (!strKey.startsWith("#")   && !strKey.trim().equals("") ) {
                                                alFileName.add(strKey) ;
                                        }
                                } catch (Exception e) {
                                }
                        }
                 configFile.close() ;

                } catch (Exception e) {
                }
        }

        public static String sFIXMsg = null ;
        public static Message fixMsg = null ;
        public static String sReportLine = null ;
        public static String   sMsgType   = null ;
        public static String sOrdType = null ;
        public static String   sSide = null ;
        public static String   sQty   = null ;
        public static String   sSymbol   = null ;
        public static String   sClOrdId = null ;
        public static String   sOrgClOrdID   = null ;
        public static String   sPx   = null ;
        public static String   sStopPx   = null ;
        public static String sTime = null ;
        public static String sExInst = null ;
        public static String sExtComp = null ;
        public static String sExtSub = null ;
        public static String sOnbehalfComp = null ;
        public static String sOnbehalfSub = null ;
        public static void parseReportOneLine(String strLine) {
                try {

                        if ((strLine.indexOf("43=Y") == -1) && (strLine.indexOf("] in.") != -1) &&
(strLine.indexOf("35=D") != -1 || strLine.indexOf("35=G") != -1)) {
//                      if ((strLine.indexOf("] out.") != -1) && strLine.indexOf("35=8") != -1 &&
(strLine.indexOf("39=1") != -1 || strLine.indexOf("39=2") != -1) /*&& (strLine.indexOf("40=2") != -1 ||
strLine.indexOf("40=3") != -1 || strLine.indexOf("40=4") != -1)*/) {
                                int iStart = strLine.indexOf("(8=FIX.") ;
                                int iLen = strLine.length() ;
                                sFIXMsg = strLine.substring(iStart+1, iLen-1) ;
                                fixMsg = new Message(sFIXMsg.getBytes()) ;

                                sExtComp = fixMsg.getStringFieldValue(Constants.TAGiSenderCompID)
;
                                sExtSub = fixMsg.getStringFieldValue(Constants.TAGiSenderSubID) ;
//                              sExtComp = fixMsg.getStringFieldValue(Constants.TAGiTargetCompID)
;
//                                      sExtSub = fixMsg.getStringFieldValue(Constants.TAGiTargetSubID) ;
                                        if (sExtSub == null) sExtSub = "NULL" ;
                                        sOnbehalfComp =
fixMsg.getStringFieldValue(Constants.TAGiOnBehalfOfCompID) ;
                                        if (sOnbehalfComp == null) sOnbehalfComp = "NULL" ;
                                        sOnbehalfSub =
fixMsg.getStringFieldValue(Constants.TAGiOnBehalfOfSubID) ;
                                        if (sOnbehalfSub == null) sOnbehalfSub = "NULL" ;
                                        sMsgType = fixMsg.getStringFieldValue(Constants.TAGiMsgType) ;
                                        sOrdType = fixMsg.getStringFieldValue(Constants.TAGiOrdType) ;
                                        sSide = fixMsg.getStringFieldValue(Constants.TAGiSide) ;
                                        sQty = fixMsg.getStringFieldValue(Constants.TAGiOrderQty) ;
//                                      sQty = fixMsg.getStringFieldValue(Constants.TAGiLastQty) ;
                                        sSymbol = fixMsg.getStringFieldValue(Constants.TAGiSymbol) ;
                                        sClOrdId = fixMsg.getStringFieldValue(Constants.TAGiClOrdID) ;
                                        if (sMsgType.equalsIgnoreCase("G")) sOrgClOrdID =
fixMsg.getStringFieldValue(Constants.TAGiOrigClOrdID) ; else sOrgClOrdID = "NULL" ;
                                        sPx = fixMsg.getStringFieldValue(Constants.TAGiPrice) ;
//                                      sPx = fixMsg.getStringFieldValue(Constants.TAGiLastPx) ;
                                        if (sOrdType!= null && sOrdType.equalsIgnoreCase("2") ) sStopPx =
"NULL"; else sStopPx = fixMsg.getStringFieldValue(Constants.TAGiStopPx) ; ;
                                        sTime = fixMsg.getStringFieldValue(Constants.TAGiSendingTime) ;
                                        sExInst = fixMsg.getStringFieldValue(Constants.TAGiExecInst) ;
                                        if (sExInst == null ) sExInst = "NULL" ;

                                        sReportLine =sTime+","+
sExtComp+","+sExtSub+","+sOnbehalfComp+","+sOnbehalfSub+","+
sMsgType+","+sOrdType+","+sSide+","+sQty+","+sSymbol+","+sClOrdId+","+sOrgClOrdID+","+sPx+","+s
StopPx+","+sExInst+"," ;

                                        bwReportOutFile.write(sReportLine+"\r\n") ;
                                        bwReportOutFile.flush() ;
                                }
                        } catch (Exception e) {
                        }
                }

                public static void generateFromOneInputFile(String sOneFileName) {
                        try {
                                BufferedReader eqListFile = new BufferedReader(new
FileReader(sOneFileName));

                                while (eqListFile.ready()) {
                                        String strKey = eqListFile.readLine().toString().trim();
                                        if (!strKey.startsWith("#")   && !strKey.trim().equals("") ) {
                                                parseReportOneLine(strKey) ;
                                        }
                                }
                         eqListFile.close() ;

                        } catch (Exception e){
                        }
                }

                public static void generateAllReportFromInputfiles() {
                        try {
```

```
                        String sOneFileName = null ;
                        int iFileNum = alFileName.size() ;
                        int iCount = 0 ;
                        while (iCount < iFileNum) {

                                sOneFileName = (String) alFileName.get(iCount) ;
                                generateFromOneInputFile(sOneFileName) ;
                                iCount++ ;
                        }
                } catch (Exception e) {
                }
        }

        public static ArrayList alFileName = new ArrayList() ;
        public static void reportGenerator() {
                try {

                        bwReportOutFile = new BufferedWriter( new FileWriter(new
File("FIX_REPORT.csv")));
                        sReportLine =
"GMTTime,Comp,Sub,OnbehalfComp,onBehalfSub,MsgType,OrdType,Side,Qty,Symbol,ClOrdID,OrgClO
rdID,Price,StopPx,ExecInst" ;

                        bwReportOutFile.write(sReportLine+"\r\n") ;
                        bwReportOutFile.flush() ;

                        loadReportInputFileNames() ;
                        generateAllReportFromInputfiles() ;
                }  catch (Exception e) {
                }
        }


    public static String sRandomType = null ;

        public static void randomOrderGenerator(String sType) {
                sRandomType = sType ;
                RandomGenerator randomGenerator = new RandomGenerator() ;
                randomGenerator.start() ;
        }

    /*
        public static void randomOrderGenerator(String sType) {
                if (sType != null && sType.toLowerCase().indexOf("ord") != -1)
                {

                        madoffDB.startFIXDatabase("dbName.txt") ;
                        loadConfig("random_order_config.txt") ;
                        createNewOutputFile("RANDOM_ORDER.csv") ;
//                      parseTitleFieldsLoc() ;
                        loadEQTDFile("ORDER_EQ_TD.csv") ;
//                      generateAllRandomOrderSequence() ;
                } else if (sType != null && sType.toLowerCase().indexOf("opt") != -1) {
                        madoffDB.startFIXDatabase("dbName.txt") ;
                        loadConfig("random_opt_config.txt") ;
                        createNewOutputFile("RANDOM_OPT.csv") ;
```

```
//				parseTitleFieldsLoc() ;
				loadEQTDFile("OPT_EQ_TD.csv") ;
//				generateAllRandomOrderSequence() ;

			}
		}
		 */
//		public static String getTradeDateNoMonth(String sDate) {
//			String sDay = "" ;
//			try {
//				int iSlashDayStart = sDate.indexOf("/") ;
//				if (iSlashDayStart != -1) sDay = sDate.substring(iSlashDayStart+1) ;
//				else {
//					int iBarDayStart = sDate.indexOf("-") ;
//					if (iBarDayStart != -1) sDay = sDate.substring(0,iBarDayStart) ;
//				}
//
//				int iSlashYearStart = sDay.indexOf("/") ;
//				if (iSlashYearStart != -1) sDay = sDay.substring(0,iSlashYearStart-1) ;
//			} catch (Exception e) {
//			}
//			return sDay ;
//		}

		public static String getTradeDateNoMonth(String sDate) {
			String sReturn = "" ;
			try {
				StringTokenizer strToken = null;
				String strElement1 = null ;
				String strElement2 = null ;
				String strElement3 = null ;
				strToken = new StringTokenizer((String) sDate, "-/ ");
				if (strToken.hasMoreTokens()) { strElement1 =   strToken.nextToken() ; if
(strElement1.length() == 4) strElement1 = strElement1.substring(2) ;}
				if (strToken.hasMoreTokens()) { strElement2 =   strToken.nextToken() ; if
(strElement2.length() == 4) strElement2 = strElement2.substring(2) ;}
				if (strToken.hasMoreTokens()) { strElement3 =   strToken.nextToken() ; if
(strElement3.length() == 4) strElement3 = strElement3.substring(2) ;	}

				if (sDate.indexOf("/") != -1) {
					sReturn =
getFixedLen(strElement2,2)+getFixedLen(strElement1,2)+getFixedLen(strElement3,2) ;
				} else if (sDate.indexOf("-") != -1) {
					sReturn =
getFixedLen(strElement3,2)+getFixedLen(strElement2,2)+getFixedLen(strElement1,2) ;
				}

		} catch (Exception e) {
		}
		return sReturn ;
	}


		public static int iMaxIDNumber = 100000000 ;	// was 100000
```

```
        public static int iMaxNumberDigit = 8 ;

        public static String getFixedLenRandomNum(String sNewID, int iMaxDigit) {
                String sNewNum = "" ;
                String sZero = "" ;
                int iIDLen = sNewID.length() ;
                int iNumZero = iMaxDigit-iIDLen ;
                for (int i = 0; i<iNumZero; i++) {
                        sZero = "0"+sZero ;
                }
                sNewNum = sZero + sNewID ;

                return sNewNum ;
        }

        public static String getFixedLen(String sNewID, int iMaxDigit) {
                String sNewNum = "" ;
                String sZero = "" ;
                int iIDLen = sNewID.length() ;
                int iNumZero = iMaxDigit-iIDLen ;
                for (int i = 0; i<iNumZero; i++) {
                        sZero = "0"+sZero ;
                }
                sNewNum = sZero + sNewID ;

                return sNewNum ;
        }


        public static String getNewID () {
                String sNewID = null ;
                try {
                        sNewID = String.valueOf(Math.abs(idGenerator.nextLong()) % iMaxIDNumber );
                        while (tmID.get(sNewID) != null ) {
                                sNewID = String.valueOf(Math.abs(idGenerator.nextLong()) %
iMaxIDNumber) ;
                        }
                        tmID.put(sNewID,"1") ;

//                      if (sNewID.length() == 1) sNewID = "0000000" + sNewID ;
//                      else if (sNewID.length() == 2) sNewID = "000000" + sNewID ;
//                      else if (sNewID.length() == 3) sNewID = "00000" + sNewID ;
//                      else if (sNewID.length() == 4) sNewID = "0000" + sNewID ;
//                      else if (sNewID.length() == 5) sNewID = "000" + sNewID ;
//                      else if (sNewID.length() == 6) sNewID = "00" + sNewID ;
//                      else if (sNewID.length() == 7) sNewID = "0" + sNewID ;



//                      if (sNewID.length() == 1) sNewID = "0000" + sNewID ;
//                      else if (sNewID.length() == 2) sNewID = "000" + sNewID ;
//                      else if (sNewID.length() == 3) sNewID = "00" + sNewID ;
//                      else if (sNewID.length() == 4) sNewID = "0" + sNewID ;

                        sNewID = getFixedLenRandomNum(sNewID, iMaxNumberDigit) ;
```

```
                              sNewID =   getTradeDateNoMonth(sSETTLE_DATE)+sNewID ;   //
getTradeDateNoMonth(sTRADE_DATE)+sNewID ;
                    } catch (Exception e) {
                    }

                    return sNewID ;
          }
}
```