```
//+------------------------------------------------------------------+
//|                        FLYER!      A Moving Average EA.mq4 |
//|                                  (GirlFlyer)  Marci Dunn |
//|                                email:  girlflyer101@yahoo.com                          |
//+------------------------------------------------------------------+

// Version 2.00;

/*  Please see accompanying documentation now for description and other information.  Thanks for using Flyer!

  */


#property copyright "Marci Dunn"
#property link      ""

//---- input parameters


// Parameters are set for EUR/USD.  Adjust for any other pairs.

extern double  LotPercentage =5;      // Percentage of equity to be used for each position.
extern double  MinVal=5000;           //Don't Let Equity drop below 5000 (example, can be set to whatever value you
want.
extern int     StopLossVal = 47;      //The stoploss value here is one key to the success rate of the EA... adjust as
needed.
                              //Suggested S/L values are
                              //EURUSD = 99, GBPUSD = 58, USDCHF = 58, USDJPY = 54
extern int     TimePeriod = 30;       // Key Timeframe to base buy/sell decisions on!!  Can be modified, but 30 min
seems to be optimal;




extern int     UseTS;                 // Do you want to use the TS?  0 = no, 1 = Yes;
extern int     TrailingStopLevel = 75;  // Trailing Stop Value;  Left in EA as an option.

extern double  BrokerMinLotSize = 0.1;  //This is the minimum lotsize your broker can accept.  If your broker uses 0.1
lots or 1 lot as
                              // minimum, set it to that value.

extern int     FixedLots = 0;         //Set to zero (0) if you want the EA to adjust your lot size automatically each trade
                              //based on the equity in your account.  Set to one (1) to trade the same size all the time.

extern double     MyLotSize = 1;          //If Fixed Lots = 1, Use this size for each lot size.  Set to the value you prefer.

extern int     LotValue = 0;          //If lots are standard value of 1 = 100000, set to zero (0).  If mini-account where lots
are 10000, set to one (1).

extern int     MarginVal = 100;        //Set your margin ratio.  Default is 100 to 1.


extern int    LimitHours = 1;         // Set to Zero to trade 24 hours a day, Set to 1 to trade only during the hours
specified.
```

```
extern int    StartHours = 21;          // TimeFrame to trade in.  Uses GMT!!!
extern int    StartMins = 0;
extern int    EndHours = 7;
extern int    EndMins = 0;
                              //*** IMPORTANT - DO NOT MODIFY ANY OPEN ORDERS OPENED BY THE EA MANUALLY AS THIS WILL LIKELY TO CAUSE PROBLEMS WITH EXECUTION

//*** Do not modify anything below ****

int       OpenOrderB;             // DO NOT MODIFY!
int       OpenOrderS;
int       OrderFlagB = 0;
int       OrderFlagS = 0;         // DO NOT MODIFY!
datetime  TimeofCross;            // DO NOT MODIFY!
int       CrossFlag = 0;          // DO NOT MODIFY!
double    PointAdjust = 0;        // Do Not Modify!
int       WaitTime = 150;         // Modify at your own risk!  Waiting Time (in seconds) for placing orders after signal valid.
                                  // Keeps transient spikes from causing orders.

int       MA = 0;
int       MA1 = 0;
int       MA2 = 0;
int       MA3,MA4,MA5;
int       MaxBars = 0;
int       GapSize = 0;            //Maximum pips price should be away from the MA line during entry process.  Keeps news events from causing
                                  //spike entries.
double    ILotVal;
int       InvalidFB,InvalidFS;
int       JustOrderedB, JustOrderedS;
int       SignalFB,SignalFS;
double    HPriceB,LPriceS,OPriceB,OPriceS;
double    PointSpreadB;
double    PointSpreadS;
int       QuickSpread;
int       TrendCnt;
double    ProfitPercent;

datetime  AlertTimer;

// For the following arrays, the data goes in the order as indicated by the SymbolList
// ie. Element #1= EUR/USD, #2 = GBP/USD, #3 = USD/CHF,#4 = USD/JPY, #5 = EUR/JPY, #6=AUDUSD
// These are the key data elements for each pair.  The performance of the EA is based upon these numbers.
// As of writing this, only EURUSD,GBPUSD,USDCHF, and USDCHF are valid.  If you wish to use any other pair, you must
// enter them and find the optimum settings.

string    SymbolList[15] = {"EURUSD","GBPUSD","USDCHF","USDJPY","EURJPY","AUDUSD"};
double    PointArray[15] = {-0.0001,-0.0001,-0.0001,-0.01,-0.01,0.0001};  //Point Array for the different symbols.
                                                  //Determines how much in pips the MA's should be crossed
                                                  //in order for the trade to be valid.
```

```
int      TrendArray[15] = {30,22,48,35,15,48,0,0,0};  //Array for the trend count length.  This factor helps determine
                                       // whether to go long or short.  It is the number of bars to go back in History
                                       // to find the current trend.  Different pairs seem to have different cycles
int      MAArray[15] = {8,4,4,8,4,3,8,8};          //Array for 1st MA.  Used in the closing of positions.
int      MA1Array[15] = {7,8,16,14,5,18,8,8};         //Short TimeFrame #1 Linear Weighted MA
int      MA2Array[15] = {9,10,18,16,7,20,10,10};    //Short TimeFrame #2 Linear Weighted MA
int      MA3Array[15] = {10,10,10,14,10,10,10,10};  //A Simple MA used in the closing of positions.  ie.. MA should
cross MA3.
int      MA4Array[15] = {18,18,40,36,20,80,18,18};  // The two long Exponential MA's.  The tunnel these two create is
"no-man's land."
                                       // The trend is indeterminate between these MA's.  Once crossed, the trend is
determined.
int      MA5Array[15] = {28,28,50,48,30,90,28,28};
int      MaxBarArray[15] = {3,5,6,8,3,6,4,4};       // How many bars on the chart can it take to cross the EMA's before
the signal becomes invalid...
int      GapArray[15] = {17,20,20,16,15,20,20,20};    // How many points can the price differ from the EMA's before it
has moved too far to be good signal.
int      ProfitArray[15] = {75,75,45,50,25,65,75,75};  //Set to the Percentage of Profit you want to take if price moves
quickly in opposite direction.
int      SpreadArray[15] = {40,58,100,40,1000,100,35,35};  //Set this level to enter a trade in the direction of motion if
the price exceeds this many pips in
                                         //15 minutes.  Can be helpful or harmful depending on the pair.  Set to a high
number if you want to
                                         //disable this feature.



string    VerString = "V200";


//+------------------------------------------------------------------+
//| expert initialization function                                   |
//+------------------------------------------------------------------+
int init()
  {
//----

  int  x;

  AlertTimer = CurTime();
  if(LotValue==0)
    {
     ILotVal = 100000 / MarginVal;
    }
  else
    if(LotValue==1)
      {
       ILotVal = 10000 / MarginVal;
      }

  for(x=0;x<15;x++)
    {
     if(StringFind(Symbol(),SymbolList[x],0)>=0)
       {
```

```
        PointAdjust = PointArray[x];
        TrendCnt = TrendArray[x];
        MA = MAArray[x];
        MA1 = MA1Array[x];
        MA2 = MA2Array[x];
        MA3 = MA3Array[x];
        MA4 = MA4Array[x];
        MA5 = MA5Array[x];
        MaxBars = MaxBarArray[x];
        GapSize = GapArray[x];
        ProfitPercent = ProfitArray[x] / 100.0;
        QuickSpread = SpreadArray[x];
      }
   }




//----
   return(0);
  }
//+------------------------------------------------------------------+
//| expert deinitialization function                                 |
//+------------------------------------------------------------------+
int deinit()
  {
//----

//----
   return(0);
  }


int  CloseCount(int  Ctr)  //Determine the trend by taking the difference between the open and the close and adding the differences.

{

  double  Diffs;

  int  x,Count;

  for(x=0,Count = 0;x<Ctr;x++)
    {

     Diffs += iClose(NULL,TimePeriod,x) - iOpen(NULL,TimePeriod,x);

    }
  Count = Diffs / Point;
  return(Count);
}
```

```
int  CloseCountMin(int  Ctr)

{

 int  x,Count;

 for(x=0,Count = 0;x<Ctr;x++)
   {
    if(iClose(NULL,1,x) >= iClose(NULL,1,x+1))
      Count++;
    else
      Count--;
   }
 return(Count);
}

void  CheckToOpenBuy(double  LotSize2)  // Checks to Open a Buy Position

 {
  int  x,flag,TicketNum;
  double  EMA18[10];
  double  EMA28[10];
  double  WMA5[10];
  double  WMA8[10];
  datetime  TimeNow;
  int     Trend;



  if(OrderFlagB>0)
    return;

  PointSpreadB = iClose(NULL,1,0)-iLow(NULL,1,15);
  for(x=0;x<MaxBars+1;x++)
    {
     EMA18[x]=iMA(NULL,TimePeriod,MA4,0,MODE_EMA,PRICE_CLOSE,MaxBars-x);
     EMA28[x]=iMA(NULL,TimePeriod,MA5,0,MODE_EMA,PRICE_CLOSE,MaxBars-x);
     WMA5[x]=iMA(NULL,TimePeriod,MA1,0,MODE_LWMA,PRICE_CLOSE,MaxBars-x);
     WMA8[x]=iMA(NULL,TimePeriod,MA2,0,MODE_LWMA,PRICE_CLOSE,MaxBars-x);
    }
  flag = 0;
  if(WMA5[MaxBars]>EMA18[MaxBars]+PointAdjust && WMA5[MaxBars]>EMA28[MaxBars]+PointAdjust &&
WMA8[MaxBars]>EMA18[MaxBars]+PointAdjust && WMA8[MaxBars]>EMA28[MaxBars]+PointAdjust)
    {
     for(x=0;x<MaxBars;x++)  //Make sure the wma's have crossed the ema's completely within recent time frame so
that flat markets don't cause trades..
        {
         if(WMA5[x]<=EMA18[x] && WMA5[x]<=EMA28[x] && WMA8[x]<=EMA18[x] &&
WMA8[x]<=EMA28[x])
           flag=1;  //Yes, they crossed within the last few periods.
        }
    }
  if(PointSpreadB > QuickSpread*Point && CloseCountMin(4)>=0)
    flag = 2;
```

```
      Trend = CloseCount(TrendCnt);
      if((flag==1 && Trend>0 && CloseCountMin(5)>=0) || flag == 2 )
        {
         if(CrossFlag==0)
           {
            TimeofCross = CurTime();
            CrossFlag = 1;
           }
         else
           {
            TimeNow = CurTime();
            if(TimeNow-TimeofCross>WaitTime)  //waiting period on orders... to prevent transient spikes in price from
causing orders.
              {
               CrossFlag = 0;
               if(Ask<EMA28[MaxBars]+GapSize*Point)  // Final check -- if price has jumped, don't order.
                 {
                  TicketNum = OrderSend(Symbol(),OP_BUY,LotSize2,Ask,3,Ask-
StopLossVal*Point,0,Symbol()+VerString,0,0,Green);  //Open the order.
                  if(TicketNum<0)
                    {
                     Print("Error Opening Trade:  Error Code = ",GetLastError(),", Lotsize: ",LotSize2);
                    }
                  else
                    {
                     if(OrderSelect(TicketNum,SELECT_BY_TICKET))
                       {
                        Print("Buy Order Opened at Price:  ",OrderOpenPrice());
                        Print("Trend Count = ",CloseCount(TrendCnt));
                        OpenOrderB = TicketNum;
                        OrderFlagB = 1;
                        InvalidFB = 0;
                        HPriceB = OrderOpenPrice();
                        OPriceB = HPriceB;
                        JustOrderedB = 1;
                       }
                    }
                 }
              }
           }
        }


  }

void  CheckToOpenSell(double  LotSize2)  //See comments in Open Buy routine for explanations.

{
   int     x,flag,TicketNum;
   double  EMA18[10];
   double  EMA28[10];
   double  WMA5[10];
   double  WMA8[10];
   datetime  TimeNow;
```

```
   int     Trend;


   if(OrderFlagS>0)
     return;
   PointSpreadS = iClose(NULL,1,0)-iHigh(NULL,1,15);
   for(x=0;x<MaxBars+1;x++)
     {
      EMA18[x]=iMA(NULL,TimePeriod,MA4,0,MODE_EMA,PRICE_CLOSE,MaxBars-x);
      EMA28[x]=iMA(NULL,TimePeriod,MA5,0,MODE_EMA,PRICE_CLOSE,MaxBars-x);
      WMA5[x]=iMA(NULL,TimePeriod,MA1,0,MODE_LWMA,PRICE_CLOSE,MaxBars-x);
      WMA8[x]=iMA(NULL,TimePeriod,MA2,0,MODE_LWMA,PRICE_CLOSE,MaxBars-x);
     }
   flag = 0;
   if(WMA5[MaxBars]<EMA18[MaxBars]-PointAdjust && WMA5[MaxBars]<EMA28[MaxBars]-PointAdjust &&
WMA8[MaxBars]<EMA18[MaxBars]-PointAdjust && WMA8[MaxBars]<EMA28[MaxBars]-PointAdjust)
     {
      for(x=0;x<MaxBars;x++)  //Check to see if the wma's have crossed the ema's recently for an entry point.
           {
            if(WMA5[x]>=EMA18[x] && WMA5[x]>=EMA28[x] && WMA8[x]>=EMA18[x] &&
WMA8[x]>=EMA28[x])
              flag=1;
           }
     }
   if(PointSpreadS < -QuickSpread*Point && CloseCountMin(4)<=0)
     flag = 2;
      Trend = CloseCount(TrendCnt);
      if((flag==1 && Trend < 0 && CloseCountMin(5)<=0) || flag ==2 )
        {
         if(CrossFlag==0)
           {
            TimeofCross = CurTime();
            CrossFlag = 1;
           }
         else
           {
            TimeNow = CurTime();
            if(TimeNow-TimeofCross>WaitTime)  //Waiting period on orders... to prevent transient spikes in price from
causing orders.
              {
               CrossFlag = 0;
               if(Bid>EMA28[MaxBars]-GapSize*Point)  // Final check -- if price has jumped, don't order.
                 {
                  TicketNum =
OrderSend(Symbol(),OP_SELL,LotSize2,Bid,3,Bid+StopLossVal*Point,0,Symbol()+VerString,0,0,Green);  //Open the
order.
                  if(TicketNum<0)
                    {
                     Print("Error Opening Trade:  Error Code = ",GetLastError(),", Lotsize: ",LotSize2);
                    }
                  else
                    {
                     if(OrderSelect(TicketNum,SELECT_BY_TICKET))
                       {
```

```
                        Print("Sell Order Opened at Price:  ",OrderOpenPrice());
                        Print("Trend Count = ",CloseCount(TrendCnt));
                        OpenOrderS = TicketNum;
                        OrderFlagS = 1;
                        InvalidFS = 0;
                        LPriceS = OrderOpenPrice();
                        OPriceS = LPriceS;
                        JustOrderedS = 1;
                      }
                   }
                }
             }
          }
       }


 }




void  CheckToOpen()  //Procedure to start the process to see if a trade can be opened.

 {
  double  LotSize;

  double  R1,R2,R3;
  double  x;
  int     flag;
  double  SmoothMASlope;
  int     TimeNowH,TimeNowM;


  if(LimitHours>0)
    {
     TimeNowH = Hour();
     TimeNowM = Minute();
     if((TimeNowH == StartHours && TimeNowM >= StartMins) || TimeNowH > StartHours)  //Are we past the
starting point of the day?
       {
        if(EndHours>=StartHours)  //Is the ending time the same day?
          {
           if((TimeNowH == EndHours && TimeNowM <= EndMins) || TimeNowH < EndHours)   //If so, are we still
within that timeframe?
             {
              return;  //Yes, return back and don't trade.
             }
          }
        else  //If the ending time is in the next day, we are automatically in the restricted time frame.  Don't trade.
          return;
       }
     else  //New day ...
       {
```

```
        if(EndHours<= StartHours && ((TimeNowH == EndHours && TimeNowM <= EndMins) || TimeNowH < EndHours))
          return;
        }
     }


   if(FixedLots==0)
     {
      LotSize = (LotPercentage/100) * AccountEquity();
      LotSize = LotSize / ILotVal;
      R1 = MathCeil(LotSize);  // Got to now round the lot size so that error 131 does not occur on some brokers.
      R2 = MathFloor(LotSize);
      for(x=R2,flag=0;x<=R1+BrokerMinLotSize && flag==0;x+=BrokerMinLotSize)
        {
         if(x>LotSize)
           {
            LotSize = x-BrokerMinLotSize;  // Round lot size to the smaller lot size value.
            flag = 1;
           }
        }
     }
   else
     {
      if(FixedLots==1)
        LotSize = MyLotSize;
     }
   if(LotSize>100)    //Check to see if lots have exceeded 100... good when account hits high enough equity to max out trade size.
      LotSize = 100;



   if(!OrderFlagB) //If we haven't got an open order, then proceed.
     {
      if(AccountEquity()>MinVal) //if equity is lower than the amount specified by user, don't trade.
        {
         if(LotSize*ILotVal<AccountFreeMargin())  //do we have enough margin to open our trade?  Should always be true unless errors.
           {
            CheckToOpenBuy(LotSize);

           }
         else
           {
            Print("Not Enough Free Margin:  ",LotSize,", ",AccountFreeMargin());
           }
        }
      else
        {
         Print("Equity Value Lower than Minimum Account Value Setting");
        }
     }
//Repeat Everything for the Sell side... necessary because an order placed on the buy size can change the equity.
```

```
   if(FixedLots==0)
     {
      LotSize = (LotPercentage/100) * AccountEquity();
      LotSize = LotSize / ILotVal;
      R1 = MathCeil(LotSize);  // Got to now round the lot size so that error 131 does not occur on some brokers.
      R2 = MathFloor(LotSize);
      for(x=R2,flag=0;x<=R1+BrokerMinLotSize && flag==0;x+=BrokerMinLotSize)
        {
         if(x>LotSize)
           {
            LotSize = x-BrokerMinLotSize;  // Round lot size to the smaller lot size value.
            flag = 1;
           }
        }
     }
   else
     {
      if(FixedLots==1)
        LotSize = MyLotSize;
     }
   if(LotSize>100)    //Check to see if lots have exceeded 100... good when account hits high enough equity to max out
trade size.
      LotSize = 100;



   if(!OrderFlagS) //If we haven't got an open order, then proceed.
     {
      if(AccountEquity()>MinVal) //if equity is lower than the amount specified by user, don't trade.
        {
         if(LotSize*ILotVal<AccountFreeMargin())  //do we have enough margin to open our trade?  Should always be
true unless errors.
           {
            CheckToOpenSell(LotSize);

           }
         else
           {
            Print("Not Enough Free Margin:  ",LotSize,", ",AccountFreeMargin());
           }
        }
      else
        {
         Print("Equity Value Lower than Minimum Account Value Setting");
        }
     }

  }

void  CheckToCloseBuy()  //Check to close any Long trades.
  {
   int  TicketNum;
   datetime  TimeOpened,Now;
```

```
   double    SMA8_1,SMA8_2;
   double    LWMA_1,LWMA_2;
   double    EMA18_1,EMA18_2,EMA28_1;

   SMA8_1 = iMA(NULL,TimePeriod,MA3,0,MODE_SMA,PRICE_CLOSE,0);
   SMA8_2 = iMA(NULL,TimePeriod,MA3,0,MODE_SMA,PRICE_CLOSE,1);
   LWMA_1 = iMA(NULL,TimePeriod,MA,0,MODE_LWMA,PRICE_CLOSE,0);
   LWMA_2 = iMA(NULL,TimePeriod,MA,0,MODE_LWMA,PRICE_CLOSE,1);
   EMA18_1 = iMA(NULL,TimePeriod,MA4,0,MODE_EMA,PRICE_CLOSE,0);
   EMA28_1 = iMA(NULL,TimePeriod,MA5,0,MODE_EMA,PRICE_CLOSE,0);
   EMA18_2 = iMA(NULL,TimePeriod,MA4,0,MODE_EMA,PRICE_CLOSE,1);



   TicketNum = OpenOrderB;
   OrderSelect(TicketNum,SELECT_BY_TICKET);
   if(Bid>HPriceB)
     HPriceB = Bid;
   if(JustOrderedB==1 && LWMA_1 < SMA8_1)
     {
      SignalFB = 1;
     }
   else
     {
      SignalFB = 0;
      JustOrderedB = 0;
     }


   if(LWMA_1< SMA8_1 && SignalFB == 0)
     {
      if(EMA18_1-EMA28_1<8*Point)
        {
         if(EMA18_1<EMA18_2-1*Point  || LWMA_1 < EMA28_1)
           {
            //if(Bid<=iClose(NULL,TimePeriod,1)) // Make sure price is going the right direction to close it
            if(CloseCountMin(5)<0)
              {
               if(Bid>OrderOpenPrice())  // Important to prevent being whipsawed all over the place, stoploss will close
losing positions.
                 {
                  CloseBuy(TicketNum);
                  OrderFlagB = 0;
                  InvalidFB = 0;
                 }
               else
                 {
                  InvalidFB = 1;
                 }
              }
           }

        }
     }
```

```
   if(HPriceB-OPriceB > 28*Point && Bid <= (OPriceB + (ProfitPercent*(HPriceB-OPriceB))) && Bid > OPriceB
&& (Bid <SMA8_1 || Bid<=OPriceB+5*Point)&& Bid<iClose(NULL,1,1) && SMA8_2 - SMA8_1 > -1.8*Point)
     {

         CloseBuy(TicketNum);
         OrderFlagB = 0;
         InvalidFB = 0;
         return;

     }
  /*  if(InvalidFB == 1 && Bid>OrderOpenPrice() && LWMA_1 < EMA18_1 && CloseCountMin()<0)  // Left in for
later possibilities...
     {
      CloseBuy(TicketNum);
      OrderFlagB = 0;
      InvalidFB = 0;
      return;
     } */

  }

void  CheckToCloseSell()  //Check to close short positions.  See CloseBuy procedure for details.
  {
   int  TicketNum;
   datetime  TimeOpened,Now;
   double    SMA8_1,SMA8_2;
   double    LWMA_1,LWMA_2;
   double    EMA18_1,EMA18_2,EMA28_1;

   SMA8_1 = iMA(NULL,TimePeriod,MA3,0,MODE_SMA,PRICE_CLOSE,0);
   SMA8_2 = iMA(NULL,TimePeriod,MA3,0,MODE_SMA,PRICE_CLOSE,1);
   LWMA_1 = iMA(NULL,TimePeriod,MA,0,MODE_LWMA,PRICE_CLOSE,0);
   LWMA_2 = iMA(NULL,TimePeriod,MA,0,MODE_LWMA,PRICE_CLOSE,1);
   EMA18_1 = iMA(NULL,TimePeriod,MA4,0,MODE_EMA,PRICE_CLOSE,0);
   EMA28_1 = iMA(NULL,TimePeriod,MA5,0,MODE_EMA,PRICE_CLOSE,0);
   EMA18_2 = iMA(NULL,TimePeriod,MA4,0,MODE_EMA,PRICE_CLOSE,1);



   TicketNum = OpenOrderS;
   OrderSelect(TicketNum,SELECT_BY_TICKET);
   if(Bid<LPriceS)
     LPriceS = Bid;

   if(JustOrderedS==1 && LWMA_1 > SMA8_1)
     {
      SignalFS = 1;
     }
   else
     {
      SignalFS = 0;
      JustOrderedS = 0;
     }
   if(LWMA_1> SMA8_1 && SignalFS == 0)     //Primary exit method
```

```
    {
     if(EMA28_1-EMA18_1<8*Point)
       {
        if(EMA18_1>EMA18_2+1*Point || LWMA_1 > EMA28_1)
          {
           //if(Ask>iClose(NULL,TimePeriod,1))
           if(CloseCountMin(5)>0)
             {
              if(Ask<OrderOpenPrice())  // Important to prevent being whipsawed all over the place, stoploss will close
losing positions.
                {
                 CloseSell(TicketNum);
                 OrderFlagS = 0;
                }
               else
                {
                 InvalidFS = 1;
                }

             }
          }
       }
    }
 /*  if(InvalidFS == 1 && Ask<OrderOpenPrice() && LWMA_1>EMA18_1 && CloseCountMin()>0)
    {
     CloseSell(TicketNum);
     OrderFlagS = 0;
     InvalidFS = 0;
     return;
    } */
   if(OPriceS-LPriceS > 28*Point && Ask >= (OPriceS - (ProfitPercent*(OPriceS-LPriceS))) && Ask < OPriceS &&
(Ask > SMA8_1 || Ask >= OPriceS-3*Point)&& Ask>iClose(NULL,TimePeriod,1) && SMA8_1-SMA8_2 > -
1.8*Point)
    {
     CloseSell(TicketNum);
     OrderFlagS = 0;
     InvalidFS = 0;
     return;
    }
 }

void  CloseBuy(int  Ticket)
 {
  if(OrderSelect(Ticket,SELECT_BY_TICKET))
    {
     if(OrderClose(Ticket,OrderLots(),Bid,3,Red))
       {
        Print("Order#",Ticket," Closed at ",OrderClosePrice());
       }
    }
  else
    {
     Print("Error Closing Ticket#",Ticket);
```

```
    }
  }

void  CloseSell(int  Ticket)
  {
   if(OrderSelect(Ticket,SELECT_BY_TICKET))
     {
      if(OrderClose(Ticket,OrderLots(),Ask,3,Red))
        {
         Print("Order#",Ticket," Closed at ",OrderClosePrice());
        }
     }
   else
     {
      Print("Error Closing Ticket#",Ticket);
     }
  }



void  CheckForStops()  // This checks to see if the open order was stopped out.  Needed to reset the open order flag.

  {
   datetime  X;

   if(OrderFlagB>0)  //  Do/Did we have an open order?
     {
      if(OrderSelect(OpenOrderB,SELECT_BY_TICKET))
        {
         X = OrderCloseTime();
         if(X>0)                 //Is there a close time?  if so, order was stopped out.
           {
            OrderFlagB = 0;  // reset open order flag.  Allows next order to be placed.
            Print("Order# ",OpenOrderB," StopLoss Close at price:  ",OrderClosePrice());
           }
        }
     }
   if(OrderFlagS>0)  //  Do/Did we have an open order?
     {
      if(OrderSelect(OpenOrderS,SELECT_BY_TICKET))
        {
         X = OrderCloseTime();
         if(X>0)                 //Is there a close time?  if so, order was stopped out.
           {
            OrderFlagS = 0;  // reset open order flag.  Allows next order to be placed.
            Print("Order# ",OpenOrderS," StopLoss Close at price:  ",OrderClosePrice());
           }
        }
     }
  }


void  CheckToClose()  //Calls the procedures to close positions if necessary.
```

```
 {
  if(OrderFlagB==1)
    {
     CheckToCloseBuy();
    }
  if(OrderFlagS==1)
    {
     CheckToCloseSell();
    }

 }

void  CheckForModify()  // Adjust Stoploss upwards as profit increases.

 {
  if(OrderFlagB==0 && OrderFlagS==0)
    return;
  //return;
  if(OrderFlagB==1)  //modify long position
    {
     if(OrderSelect(OpenOrderB,SELECT_BY_TICKET))
       {
        if(OrderStopLoss()<OrderOpenPrice())
          {
           if(Bid>OrderOpenPrice()+55*Point)
             {

if(OrderModify(OpenOrderB,OrderOpenPrice(),OrderOpenPrice()+15*Point,OrderOpenPrice()+700*Point,0,Yellow)==
=false)
                 {
                  Print("Error Modifying Order! Code: ",GetLastError());
                 }
             }
           else

           if(Bid>OrderOpenPrice()+TrailingStopLevel*Point && UseTS == 1)
             {

if(OrderModify(OpenOrderB,OrderOpenPrice(),OrderOpenPrice()+10*Point,OrderOpenPrice()+700*Point,0,Yellow)==
=false)
                 {
                  Print("Error Modifying Order! Code: ",GetLastError());
                 }

             }
          }
        else
          {
           if(Bid>(OrderStopLoss()+(TrailingStopLevel*Point*2)) && UseTS == 1)
             {

if(OrderModify(OpenOrderB,OrderOpenPrice(),OrderStopLoss()+(TrailingStopLevel)*Point,OrderStopLoss()+700*Po
int,0,Yellow)==false)
                 {
```

```
                  Print("Error Modifying Order! Code: ",GetLastError());
                 }
               }
             }
           }
       }
   if(OrderFlagS==1)  // modify short position
     {
      if(OrderSelect(OpenOrderS,SELECT_BY_TICKET))
        {
         if(OrderStopLoss()>OrderOpenPrice())
           {
            if(Ask<OrderOpenPrice()-55*Point)
              {
               if(OrderModify(OpenOrderS,OrderOpenPrice(),OrderOpenPrice()-15*Point,OrderOpenPrice()-
700*Point,0,Yellow)==false)
                  Print("Error Setting S/L to 5",GetLastError());

              }
            else

            if(Ask<OrderOpenPrice()-TrailingStopLevel*Point && UseTS == 1)
              {
               if(OrderModify(OpenOrderS,OrderOpenPrice(),OrderOpenPrice()-10*Point,OrderOpenPrice()-
700*Point,0,Yellow)==false)
                  Print("Error Setting S/L to 5",GetLastError());

              }

           }
         else
           {
            if(Ask<(OrderStopLoss()-(TrailingStopLevel*Point*2)) && UseTS == 1)
              {
               if(OrderModify(OpenOrderS,OrderOpenPrice(),OrderStopLoss()-
(TrailingStopLevel)*Point,OrderStopLoss()-700*Point,0,Yellow)==false)
                 {
                  Print("Error Modifying Order! Code: ",GetLastError());
                 }
              }
           }
        }
     }
  }

void CheckForOpenOrders()

  {
   //This routine becomes necessary in the event of power failures, program interruption, or chart change.

   int  x,TotalOrders;

   TotalOrders = OrdersTotal();
   for(x=0;x<TotalOrders;x++)
```

```
    {
     if (OrderSelect(x, SELECT_BY_POS, MODE_TRADES))
       {
        if(StringFind(OrderComment(),Symbol()+VerString,0)>=0)
          {
           if(OrderType()==OP_BUY)
             {
              OrderFlagB = 1;
              OpenOrderB = OrderTicket();
             }
           else
            if(OrderType()==OP_SELL)
              {
               OrderFlagS = 1;
               OpenOrderS = OrderTicket();
              }
          }
       }
    }
  }

//+------------------------------------------------------------------+
//| expert start function                                            |
//+------------------------------------------------------------------+
int start()
  {
//----
   HideTestIndicators(false);
   CheckForOpenOrders();
   CheckToOpen();
   CheckToClose();
   CheckForModify();
   CheckForStops();
if(CurTime()>=AlertTimer+300)
    {
     AlertTimer += 300;
     //Print("Account Equity = ",AccountEquity()," Bid = ",Bid," Ask = ",Ask," OrderStat = ",OrderFlag);
    }
   if(OrderFlagB>0 || OrderFlagS>0)
     Comment("Open Orders Exist, Equity = ",AccountEquity());
   else
     Comment("No Open Orders! Waiting for Entry.");

//----
   return(0);
  }
//+-
```