```
//+------------------------------------------------------------------+
//|                               EA_BasicStrategy_Adjust1_Optimizer.mq4 |
//|                                            Copyright 2017, M Wilson |
//|                                         https://www.algotrader.blog |
//+------------------------------------------------------------------+
#include <C_TradeManagement.mqh>
#include <C_OPTIMIZATION_LOG.mqh>

#property copyright "Copyright 2017, M Wilson"
#property link      "https://www.algotrader.blog"
#property version   "1.00"
#property strict

//+------------------------------------------------------------------+
//| Inputs                                                           |
//+------------------------------------------------------------------+
input int I_MagicNumber = 20170302;
input double I_RiskRewardRatio=1.5;
//Size of Take Profit relative to StopLoss.
input int I_Slippage=5;                          //Slippage for Trading.
input int I_MinimumStoplossToTradeInPoints=30;
//No Trading if the stoploss is less than this.
input double I_StoplossRiskInAcctCurrency=100;
//The amount to risk in the currency of the trading account.
input double I_MaxLotSize=0.5;
//To restrict GAP Risk, Lot Size is capped at this number.
input bool I_UseOptimizationLog=False;           //Write to the Optimization Log File

//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
datetime g_dtLastCheck;
C_TradeManagement *g_TradeManagement;
C_OPTIMIZATION_LOG *g_optLog;

//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
   g_TradeManagement = new C_TradeManagement(I_MagicNumber);

   //If we are in the testing environment, then initiate the optimization log
   if(I_UseOptimizationLog)   g_optLog = new C_OPTIMIZATION_LOG(I_MagicNumber,Symbol());

   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {

    //If we are in the testing environment, then initiate the optimization log
   if(I_UseOptimizationLog)
   {
      string strParameters=GetEAParameters();
      g_optLog.WriteSummaryToLogFile(strParameters);
      g_optLog.PrintLocationOfLogFiles();
      if(g_optLog!=NULL)   delete g_optLog;
   }

   if(g_TradeManagement!=NULL)   delete g_TradeManagement;

  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {


//This section of code ensures that anything following it is only run once per candle.   It
```

```
//store the time of the last run and compares this to the time in candle 1.   When the time
   //be large enough to run the routine.
   if(MathAbs(g_dtLastCheck-iTime(Symbol(),0,1))<(PeriodSeconds()/2))   return;
   g_dtLastCheck=iTime(Symbol(),0,1);

   //CODE THAT IS RUN ONCE PER CANDLE ...

   //If there are no live trades, then check to see if we open a new trade.
   if(g_TradeManagement.CountLiveTrades()<1)
   {


//This is a simple strategy, if the previous candle was up, then we buy, if the previous car

//then we sell.   There are restrictions on the side of the stoploss (I_ExtraSpreadToTrade)
      //trade.

      double dblStopLoss, dblTakeProfit;
      bool boolAddTrade=False;
      ENUM_ORDER_TYPE eOrderType;

      if(iClose(Symbol(),0,1)>iOpen(Symbol(),0,1))
      {
         eOrderType=OP_BUY;

         //Put the stoploss behind the Low of the previuos candle
         dblStopLoss=iLow(Symbol(),0,1);
         dblTakeProfit=Ask+((Ask-dblStopLoss)*I_RiskRewardRatio);

         //Normalize the values
         dblStopLoss=NormalizeDouble(dblStopLoss,Digits);
         dblTakeProfit=NormalizeDouble(dblTakeProfit,Digits);

         CreateOrder(eOrderType,dblStopLoss,dblTakeProfit);

         boolAddTrade=True;
      }
      else if(iClose(Symbol(),0,1)<iOpen(Symbol(),0,1))
      {
         eOrderType=OP_SELL;

         //Put the stoploss above the high of the previous candle
         dblStopLoss=iHigh(Symbol(),0,1);
         dblTakeProfit=Bid-((dblStopLoss-Bid)*I_RiskRewardRatio);

         //Normalize the values
         dblStopLoss=NormalizeDouble(dblStopLoss,Digits);
         dblTakeProfit=NormalizeDouble(dblTakeProfit,Digits);

         CreateOrder(eOrderType,dblStopLoss,dblTakeProfit);

      }
   }


//This part provides the earliest and latest date for the optimization log.   In the opt log

//the input is less than the current value, so running this every tick covers the full date
   if(I_UseOptimizationLog)
   {

//Each pass of the optimization log, we need to record the minimum start date and the last e
      g_optLog.StartDate(iTime(Symbol(),0,1));
      g_optLog.EndDate(iTime(Symbol(),0,0));
   }

  }

//+------------------------------------------------------------------+
//| Trade Management Functions                                       |
//+------------------------------------------------------------------+
```

```
int CreateOrder(const ENUM_ORDER_TYPE eTradeDirection, const double dblStopLoss=0, const
 double dblTakeProfit=0)
{

   //Define Constants.
   bool boolContinue=True;
   int intErr=0, intTicket=-1;
   string strBrokerXML="";


//Function attempts to create a trade of the type specified by eBUYorSELL and returns the ti
   //number <=0 if it fails

   RefreshRates();

   //Define integer used by OrderSend to define Buy or Sell
   double dblSpot=Ask;
   color clrTradeDirection = clrLightGreen;
   if(eTradeDirection==OP_SELL)
   {
      dblSpot=Bid;
      clrTradeDirection=clrLightPink;
   }


//Ensure that the stoploss is outside of the I_MaximumStoplossToTradeInPoints before initiat
   if(MathAbs(dblSpot-dblStopLoss)<I_MinimumStoplossToTradeInPoints*Point())
   {
      Print(__FILE__+" : "+__FUNCTION__," ",TimeCurrent(),
" StopLoss is too close to the spot to trade");
      return -1;
   }


//Ensure that the Bid/Ask spread is greater than the StopLoss by at least the slippage other
   if(MathAbs(dblSpot-dblStopLoss)-I_Slippage*Point()<MathAbs(Ask-Bid))
   {
      Print(__FILE__+" : "+__FUNCTION__," ",TimeCurrent(),
" Bid/Ask spread too wide to trade.");
      return -1;
   }

   //Get the risk for 1 lot, this will exclude commission and swap rates etc etc.
   double dblAtRisk1Lot=g_TradeManagement.CalculateAtRisk1Lot(MathAbs(dblSpot-
dblStopLoss));


//Calculate the Lot size based upon our inputs (I have left the 1.0 as a visual reminder of
   double dblLotSize=1.0*I_StoplossRiskInAcctCurrency/dblAtRisk1Lot;

   //Ensure we do not exceed the maximum lot size
   if(dblLotSize>I_MaxLotSize)
   {
      dblLotSize=I_MaxLotSize;
      if(I_UseOptimizationLog) g_optLog.IncrementNoGAPRiskLimitedTrades();
   }

   //Round the Lot Size to ensure that it is tradable
   dblLotSize=g_TradeManagement.RoundLotSize(dblLotSize);

   //Only attempt to trade if there will be enough free margin
   ResetLastError();
   if(AccountFreeMarginCheck(Symbol(),eTradeDirection,dblLotSize)<0 || GetLastError()==
134)
   {
      Print(__FILE__+" : "+__FUNCTION__," ",TimeCurrent(),
" Not enough Free Margin to Trade");
      return -1;
   }

   //Reset any errors
   ResetLastError();
```

```
   //Attempt to add the trade up to 5 times
   for(int i=0;i<5;i++)
   {
      RefreshRates();

      //Keep refreshing the spot while looping
      dblSpot=Ask;
      if(eTradeDirection==OP_SELL) dblSpot =Bid;

      //Attempt to open a trade
      intTicket=OrderSend(Symbol(),eTradeDirection,dblLotSize,dblSpot,I_Slippage,
dblStopLoss,dblTakeProfit,"",I_MagicNumber,0,clrTradeDirection);
      if(intTicket>0)
      {  //Ticket Successfully added - potentially process here.


//If we successfully add a trade, increment the amount at risk recorded in the optimization
         if(I_UseOptimizationLog)
         {
            if(OrderSelect(intTicket,SELECT_BY_TICKET,MODE_TRADES))
            {
               double dblOptimizerAtRisk1Lot=g_TradeManagement.CalculateAtRisk1Lot(
MathAbs(OrderOpenPrice()-OrderStopLoss()));
               double dblOptimizerAtRisk=OrderLots()*dblOptimizerAtRisk1Lot;
               g_optLog.AddToAmountAtRisk(dblOptimizerAtRisk);
            }
         }

         //Break out of the routine.
         break;
      }
      else
      {  //Error Adding the Trade.
         intErr = GetLastError();
         boolContinue = g_TradeManagement.DoWeContinueAttemptingToTrade(intErr);
         if(!boolContinue) break;
      }

   }

   //Report any errors if the trade was not added successfully.
   if(intTicket<=0)
   {
      //Problem creating the trade - report errors using print and on the chart report
      if(boolContinue)
      {
         Print(__FUNCTION__,
" TRADE ENTRY ERROR, COULD NOT OPEN TRADE AFTER 5 ATTEMPTS (SEE LOG): ",intErr);
      }
      else
      {
         Print(__FUNCTION__," CRITICAL TRADE ENTRY ERROR (SEE LOG): ",intErr);
      }
   }

   return intTicket;
}
//+------------------------------------------------------------------+
//| General EA Functions                                             |
//+------------------------------------------------------------------+
string GetEAParameters()
{
   string strRet="I_MagicNumber:"+IntegerToString(I_MagicNumber);
   strRet+=",I_RiskRewardRatio:"+DoubleToString(I_RiskRewardRatio,Digits());
   strRet+=",I_Slippage:"+IntegerToString(I_Slippage);
   strRet+=",I_MinimumStoplossToTradeInPoints:"+IntegerToString(
I_MinimumStoplossToTradeInPoints);
   strRet+=",I_StoplossRiskInAcctCurrency:"+DoubleToString(I_StoplossRiskInAcctCurrency,
Digits());
   strRet+=",I_MaxLotSize:"+DoubleToString(I_MaxLotSize,Digits());
   strRet+=",I_UseOptimizationLog:"+IntegerToString(I_UseOptimizationLog);

   return strRet;
```

```
}
```