

```

//+-----+
//|                                     EA_BasicStrategy_Adjust3_Calendar.mq4 |
//|                                     M Wilson |
//|                                     https://www.algotrader.blog |
//+-----+
#include <C_TradeManagement.mqh>
#include <C_OPTIMIZATION_LOG.mqh>
#include <C_Calendar_ForexFactoryImport.mqh>

#property copyright "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;
input int I_Slippage=5;
input int I_MinimumStoplossToTradeInPoints=30;
input double I_StoplossRiskInAcctCurrency=100;
input double I_MaxLotSize=0.5;
input int I_MaximumSpreadToTradeInPoints=-1;
input int I_TradingStartHour=6;
input int I_TradingEndHour=21;
input int I_SpreadWideningStartHour=21;
input int I_SpreadWideningEndHour=0;
input int I_SWCloseDistanceInPoints=90;
input int I_WeekdayCloseHour=-1;
input int I_WeekendCloseHour=19;
input int I_CalendarTimeOffsetInMinutes=0;
input int I_CalendarMinutesBefore=10;
input int I_CalendarMinutesAfter=20;
input bool I_ProcessAllDayEvents=False;
input int I_ExpandAllDayEventsByXHours=0;
input bool I_UseOptimizationLog=False;

//+-----+
//| Global Variables |
//+-----+
datetime g_dtLastTradeEntryCheck;
datetime g_dtLastTradeManagementCheck;
datetime g_dtLastMinute;
C_TradeManagement *g_TradeManagement;
C_Calendar_ForexFactoryImport *g_Calendar;
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());

    //Validate I_WeekdayCloseHour
    if(I_WeekdayCloseHour>=0)
    {
        if(I_TradingStartHour>I_TradingEndHour)
        {
            if(I_WeekdayCloseHour>=I_TradingStartHour || I_WeekdayCloseHour<
I_TradingEndHour)
            {
                Print(__FILE__+" : "+__FUNCTION__,
" Weekday Close Hour cannot be during the trading day");
                return(INIT_FAILED);
            }
        }
    }
    else

```

```

        {
            if(I_WeekdayCloseHour>=I_TradingStartHour && I_WeekdayCloseHour<
I_TradingEndHour)
            {
                Print(__FILE__+" : "+__FUNCTION__,
" Weekday Close Hour cannot be during the trading day");
                return(INIT_FAILED);
            }
        }
    }

    //Get an array of countries that we are interested in
    string strCountries[];
    populateCountryArray(strCountries);

    g_Calendar = new C_Calendar_ForexFactoryImport(strCountries,False,False,True,
I_CalendarTimeOffsetInMinutes*60);
    if(IsTesting() || IsOptimization())
    {
//We are in the testing or optimization environment, sync the calendar once at the start wit
    g_Calendar.SyncCalendar("ffcal_week_history.xml");
    }

    ArrayFree(strCountries);

    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=GetEAPParameters();
        g_optLog.WriteSummaryToLogFile(strParameters);
        g_optLog.PrintLocationOfLogFiles();
        if(g_optLog!=NULL) delete g_optLog;
    }

    if(g_TradeManagement!=NULL) delete g_TradeManagement;

    if(g_Calendar!=NULL) delete g_Calendar;
}
//+-----+
//| Expert tick function |
//+-----+
void OnTick()
{
//This section of code is designed to run maintenance code once per minute when we are not r
    datetime dtMinStartTime=iTime(Symbol(),0,0)+20;
    datetime dtMaxStartTime=iTime(Symbol(),0,0)+PeriodSeconds()-20;
    if(dtMinStartTime>=dtMaxStartTime) Print(__FILE__+" : "+__FUNCTION__,
" ERROR Running once per minute");
    if(TimeCurrent()>dtMinStartTime && TimeCurrent()<dtMaxStartTime && MathAbs(
TimeCurrent()-g_dtLastMinute)>60)
    {
//This code is run once per minute as long as we are not within 20 seconds of the start or e

        //Close any trades that are open during the weekday close hour if set.
        CloseWeedayTrades();

        //Close any trades that are open during the pre weekend close hour.
        CloseWeekendTrades();
    }
}

```

```

//If Close At End of Day is set, then close trades.   Can also check to close trades if they
    CloseTradesCloseToStopLoss();

//Every minute, in the live environment, check the time of the calendar file and the time of
//they are out by more than 1 minute, then re-sync the calendar.
    if(!IsTesting() && !IsOptimization())
    {
        int intSecDiff = MathAbs((int)(g_Calendar.GetLastSyncDate()-
g_Calendar.GetLastModifyDateFromFile("ffcal_week_this.xml")));
        if(intSecDiff>30) g_Calendar.SyncCalendar("ffcal_week_this.xml");
    }

    g_dtLastMinute=TimeCurrent();
}

//This section of code is run once per candle, however if there is a calendar event near buy
//so the we do not process the trade entry code AND that once-per-candle routine until the e
//For example if you have a calendar event at 10:00 and you are using the hourly chart, the
//10:20, rather than 10:00, 11:00, 12:00 etc
    int intTimeDiff=(int)(g_dtLastTradeEntryCheck-iTime(Symbol(),0,1));
    intTimeDiff=MathAbs(intTimeDiff);
    if(intTimeDiff > PeriodSeconds()/2)
    {
        if(CanWeProcessTradeCandles(TimeCurrent()))
        {

//Update the global variable, so that this section of code will not be run again until the r
    g_dtLastTradeEntryCheck=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.   We cannot do this
//or when we are past the weekend.
        if(CanWeCreateNewTrades())
        {
            LookForNewTrade();
        }
    }

//This section of code is run once per candle.
//It is used for trade management, such as moving stoplosses, not for trade entry.
    int intTimeDiffMgmt=(int)(g_dtLastTradeManagementCheck-iTime(Symbol(),0,1));
    intTimeDiffMgmt=MathAbs(intTimeDiff);
    if(intTimeDiffMgmt > PeriodSeconds()/2)
    {

//Update the global variable, so that this section of code will not be run again until the r
    g_dtLastTradeManagementCheck=iTime(Symbol(),0,1);

        //Manage Trades, eg close or stoploss movements.
        ManageTrades();

//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 |
//+-----+
void LookForNewTrade()
{
//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);

}
}
void ManageTrades()
{
}
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 initiating
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 otherwise
if(MathAbs(dblSpot-dblStopLoss)-I_Slippage*Point()<MathAbs(Ask-Bid))
{
    Print(__FILE__+" : "+__FUNCTION__, " ",TimeCurrent(),
" Bid/Ask spread too wide to trade.");
    return -1;
}

//Ensure that the spot-stoploss is greater than the StopLevel
double dblStopLevel=MarketInfo(Symbol(),MODE_STOPLEVEL)*MarketInfo(Symbol(),
MODE_POINT);
double dblFreezeLevel=MarketInfo(Symbol(),MODE_FREEZELEVEL)*MarketInfo(Symbol(),
MODE_POINT);
if(MathAbs(dblSpot-dblStopLoss)>=dblStopLevel)
{
    Print(__FILE__+" : "+__FUNCTION__,
" StopLevel is greater than spot - stoploss. Trade may not work.");
}
else if(MathAbs(dblSpot-dblStopLoss)>=dblFreezeLevel)
{
    Print(__FILE__+" : "+__FUNCTION__,
" FreezeLevel is greater than spot - stoploss. Trade may not work.");
}

//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;
}
bool CloseOrder(const int intTicket=-1)
{
    bool boolRes = True, boolContinue=True;
    color clrTradeDirection;

    //Do not process unless we have a valid ticket no.
    if(intTicket<0) return False;

    if(OrderSelect(intTicket,SELECT_BY_TICKET,MODE_TRADES))
    {
        //Check if symbol and magic number match

```

```

        if(Symbol()==OrderSymbol() && I_MagicNumber==OrderMagicNumber() && (OrderType()==
OP_BUY || OrderType()==OP_SELL))
    {
        //Reset any errors
        ResetLastError();

        //Attempt to close the trade 10 times.
        for(int j=0;j<10;j++)
        {
            RefreshRates();

//Get the rate to close the trade at (sell buy order-Bid, buy sell order - ask).
            double dblSpot=Bid;
            clrTradeDirection=clrGreen;
            if(OrderType()==OP_SELL)
            {
                dblSpot=Ask;
                clrTradeDirection=clrPink;
            }

//Attempt to close the order. If there are any errors decide if we wish to continue attempt
            boolRes=OrderClose(OrderTicket(),OrderLots(),dblSpot,I_Slippage,
clrTradeDirection);
            if(boolRes) break;
            boolContinue=g_TradeManagement.DoWeContinueAttemptingToTrade(GetLastError
());
            if(!boolContinue) break;
        }

        //Report any errors attempting to close the order
        if(!boolRes)
        {
            if(boolContinue)
            {
                Print(__FUNCTION__,
"TRADE CLOSE ERROR, COULD NOT CLOSE TRADE AFTER 10 ATTEMPTS: ",GetLastError());
            }
            else
            {
                Print(__FUNCTION__, "CRITICAL TRADE CLOSE ERROR: ",GetLastError());
            }
        }
    }
}
else
    Print(__FUNCTION__, "TRADE ERROR - COULD NOT SELECT TRADE: ",GetLastError());

return boolRes;
}
bool CloseAllOrders()
{
    bool boolRes = True;

//This function closes all open orders (not limit/stop)

//Get the number of open trades
int intNoTrades=OrdersTotal();

//Scan through the open orders
for(int i=intNoTrades-1;i>=0;i--)
{
    if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES))
    {
        int intTicket=OrderTicket();

        //Attempt to close the order.
        boolRes = (boolRes && CloseOrder(intTicket));
    }
}
}

```

```

        else
            Print(__FUNCTION__, "TRADE ERROR - COULD NOT SELECT TRADE: ", GetLastError());
    }

    return boolRes;
}
//+-----+
//| 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_MaximumSpreadToTradeInPoints:"+IntegerToString(
I_MaximumSpreadToTradeInPoints);
    strRet+="I_TradingStartHour:"+IntegerToString(I_TradingStartHour);
    strRet+="I_TradingEndHour:"+IntegerToString(I_TradingEndHour);
    strRet+="I_SpreadWideningStartHour:"+IntegerToString(I_TradingStartHour);
    strRet+="I_SpreadWideningEndHour:"+IntegerToString(I_TradingEndHour);
    strRet+="I_SWCloseDistanceInPoints:"+IntegerToString(I_SWCloseDistanceInPoints);
    strRet+="I_WeekdayCloseHour:"+IntegerToString(I_WeekdayCloseHour);
    strRet+="I_WeekendCloseHour:"+IntegerToString(I_WeekendCloseHour);
    strRet+="I_CalendarTimeOffsetInMinutes:"+IntegerToString(
I_CalendarTimeOffsetInMinutes);
    strRet+="I_CalendarMinutesBefore:"+IntegerToString(I_CalendarMinutesBefore);
    strRet+="I_CalendarMinutesAfter:"+IntegerToString(I_CalendarMinutesAfter);
    if(I_ProcessAllDayEvents)
    {
        strRet+="I_ProcessAllDayEvents:True";
    }
    else
    {
        strRet+="I_ProcessAllDayEvents:False";
    }
    strRet+="I_ExpandAllDayEventsByXHours:"+IntegerToString(I_ExpandAllDayEventsByXHours
);
    strRet+="I_UseOptimizationLog:"+IntegerToString(I_UseOptimizationLog);

    return strRet;
}
bool CanWeCreateNewTrades()
{
    bool boolRet = True;

    //Cannot trade if we have any live trades.
    if(g_TradeManagement.CountLiveTrades(>0) boolRet=False;

    //Evaluate if we are in the weekend - we do not trade on saturday or sunday.
    int intDay=TimeDayOfWeek(TimeCurrent());
    if(intDay==0 || intDay==6) boolRet=False;

    //Evaluate if we are within the trading day.    If the start hour is greater than the end hou
    //we are day trading.

    //PLEASE NOTE THAT WHEN YOU RUN THE STRATEGY SIMULATOR, IT MAKES THE LOCAL TIME AND CURRENT
    //TO BUILD A STRATEGY BASED AROUND CURRENTTIME RATHER THAN LOCALTIME.
    int intCurrHour = TimeHour(TimeCurrent());
    if(I_TradingStartHour>I_TradingEndHour)
    {

    //If we are NOT (greater than or equal to the trading start hour OR less than the End tradir
    if(!(intCurrHour>=I_TradingStartHour || intCurrHour<I_TradingEndHour))    boolRet=
False;
    }
}

```



```

else
{
//If we are NOT (greater than or equal to the trading start hour and less than the End Tradi
    if(!(intCurrHour>=I_TradingStartHour && intCurrHour<I_TradingEndHour))    boolRet=
False;
}

//Ensure we cannot trade at any point in time greater than or equal to the weekend close hou
    if(IsPreWeekendDay())
    {
        if(intCurrHour>=I_WeekendCloseHour) boolRet=False;
    }

//We do not create a new trade on bank holidays.    Do not auto-include the US in the countri
    string strCountries[];
    populateCountryArray(strCountries,False,False);
    int intSize=ArraySize(strCountries);
    for(int i=0;i<intSize;i++)
    {
        string strC=strCountries[i];
        if(StringLen(strC)>0)
        {
            if(strC=="EUR")
            { //Only count EUR bank holidays that include France and Germany.
                bool boolFrench=g_Calendar.TodayIsBankHoliday(TimeCurrent(),strC,"France")
|| g_Calendar.TodayIsBankHoliday(TimeCurrent(),strC,"French");
                bool boolGerman=g_Calendar.TodayIsBankHoliday(TimeCurrent(),strC,"German");
                if(boolFrench && boolGerman)
                {
                    boolRet=False;
                    break;
                }
            }
            else
            {
                if(g_Calendar.TodayIsBankHoliday(TimeCurrent(),strC,""))
                {
                    boolRet=False;
                    break;
                }
            }
        }
    }
    ArrayFree(strCountries);

    return boolRet;
}
bool CanWeProcessTradeCandles(const datetime dtCurrent=0)
{
//When we have the calendar, we may want to prevent trading say 10 minutes before an event a
//are say on the 1 hour timeframe and the event is at 10:00 we don't want to stop trading ur
//new trades being added until 10:10.

//This routine tells us if we are in a period of time that is important for calendar events.

    bool boolRet=True;

//We do not want to trade if the spread is too large.
    if(I_MaximumSpreadToTradeInPoints>0)
    {
        double dblSpread = Ask-Bid;
        double dblMax=I_MaximumSpreadToTradeInPoints*MarketInfo(Symbol(),MODE_POINT);
        if(dblSpread>dblMax) return False;
    }
}

```

```

//If I do not want to trade 10 minutes before a calendar event, and 20 minutes after a caler
//time and see if there are any events between now+10 minutes and now-20 minutes. Hence Af
//than you would expect.
datetime dtStart=dtCurrent-(I_CalendarMinutesAfter*60);
datetime dtEnd=dtCurrent+(I_CalendarMinutesBefore*60);
NewsImpact eImpact;

if(g_Calendar.HighestImpactEventBetweenDates(dtStart,dtEnd,eImpact))
{
    if(eImpact==HighImpact) return False;
}

if(I_ProcessAllDayEvents)
{
    datetime dtStartAD=dtStart+(I_ExpandAllDayEventsByXHours*60*60);
    datetime dtEndAD=dtStart-(I_ExpandAllDayEventsByXHours*60*60);
    NewsImpact eStartImpact, eEndImpact;

    if(g_Calendar.HighestImpactAllDayEvent(dtStartAD,eStartImpact) ||
g_Calendar.HighestImpactAllDayEvent(dtEndAD,eEndImpact))
    {
        if(eStartImpact==HighImpact || eEndImpact==HighImpact) return False;
    }
}

return boolRet;
}
void CloseWeedayTrades()
{
    if(I_WeekdayCloseHour<0 || I_WeekdayCloseHour>23) return;

    int intCurrHour=TimeHour(TimeCurrent());
    if(intCurrHour==I_WeekdayCloseHour)
    {
        if(g_TradeManagement.CountLiveTrades(>0)
        {
            //Need Code to close all open orders.
            CloseAllOrders();
        }
    }
}
void CloseWeekendTrades()
{
    if(g_TradeManagement.CountLiveTrades(>0)
    {
        if(IsPreWeekendDay())
        {
            int intCurrHour=TimeHour(TimeCurrent());
            if(intCurrHour>=I_WeekendCloseHour)
            {
                //Need Code to close all open orders.
                CloseAllOrders();
            }
        }
    }
}
void CloseTradesCloseToStopLoss()
{
    //Do not process this routine if I_SWCloseDistanceInPoints is less than or equal to zero.
    //distance to the stoploss is less than or equal to zero are closed, which doesn't make any
    //less than or equal to zero is a way of turning this feature off.
    if(I_SWCloseDistanceInPoints<=0) return;

    bool boolTimeForCheck=False;
    int intCurrHour = TimeHour(TimeCurrent());
    if(I_SpreadWideningStartHour>I_SpreadWideningEndHour)
    {

```

```

//If we are (greater than or equal to the trading start hour OR less than the End trading hour)
    if(intCurrHour>=I_SpreadWideningStartHour || intCurrHour<I_SpreadWideningEndHour)
        boolTimeForCheck=True;
    }
    else
    {

//If we are (greater than or equal to the end trading hour or less than the start Trading hour)
    if((intCurrHour>=I_SpreadWideningStartHour && intCurrHour<I_SpreadWideningEndHour)
)    boolTimeForCheck=True;
    }

//Exit if we do not need to check.
if(!boolTimeForCheck)    return;

int intNoTrades=g_TradeManagement.CountLiveTrades();
if(intNoTrades>0)
{

//Scan through the open orders and close those which are close to the stoploss.    This is us
//in spreads that you get between 22:00 and 23:00 which can result in trades closing.
for(int i=intNoTrades-1;i>=0;i--)
{
    if(OrderSelect(i,SELECT_BY_POS))
    {
        //Work out how close we are to the spot.
        double dblDistance=MathAbs(Bid-OrderStopLoss());
        if(OrderType()==OP_SELL)    dblDistance=MathAbs(OrderStopLoss()-Ask);
        //Get the distance for us to close.
        double dblMaxDist=I_SWCloseDistanceInPoints*MarketInfo(Symbol(),MODE_POINT);
        //Get the ticket
        int intTicket=OrderTicket();
        //Attempt to close the order if appropriate.
        if(dblDistance<dblMaxDist)
        {
            //Close the order.
            if(CloseOrder(intTicket))    Print(__FUNCTION__,
" Closed Order, End-Of-Day, too close to StopLoss.    Tkt: ",intTicket);
        }
        else
            Print(__FUNCTION__, " Could not select trade by position.    Pos No: ",i,
" ErrNo:",GetLastError());
    }
}
}
bool IsPreWeekendDay()
{
    bool boolRet=False;

//0 is sunday, 5 is Friday
if(TimeDayOfWeek(TimeCurrent())==5) boolRet=True;

    return boolRet;
}
void populateCountryArray(string &strCountries[], const bool boolAddUSIfMissing=True,
const bool boolAddALLCountry=True)
{

//We are going to ensure that we have the account currency, the base currency, the profit cu
//in the array of countries.

bool boolAddUS=True;
int intSize=ArraySize(strCountries);

string strAccount=StringTrimLeft(StringTrimRight(AccountCurrency()));
StringToUpper(strAccount);
if(strAccount=="USD")    boolAddUS=False;
ArrayResize(strCountries,intSize+1);

```

```

strCountries[intSize]=strAccount;
intSize++;

string strBase=StringTrimLeft(StringTrimRight(SymbolInfoString(Symbol(),
SYMBOL_CURRENCY_BASE)));
StringToUpper(strBase);
if(strBase=="USD")    boolAddUS=False;
if(strBase!=strAccount)
{
    ArrayResize(strCountries,intSize+1);
    strCountries[intSize]=strBase;
    intSize++;
}

string strProfit=StringTrimLeft(StringTrimRight(SymbolInfoString(Symbol(),
SYMBOL_CURRENCY_PROFIT)));
StringToUpper(strProfit);
if(strProfit=="USD")    boolAddUS=False;
if(strProfit!=strAccount && strProfit!=strBase)
{
    ArrayResize(strCountries,intSize+1);
    strCountries[intSize]=strProfit;
    intSize++;
}

if(boolAddUS && boolAddUSIfMissing)
{
    ArrayResize(strCountries,intSize+1);
    strCountries[intSize]="USD";
    intSize++;
}

if(boolAddALLCountry)
{
    ArrayResize(strCountries,intSize+1);
    strCountries[intSize]="ALL";
    intSize++;
}
}

```