//+------------------------------------------------------------------+

//| Expert initialization function |

//+------------------------------------------------------------------+

#property strict

input double RiskPercent = 1.0; // Risk per trade as a percentage of account balance

input int TakeProfitFactor = 25; // Reward-to-risk ratio (e.g., 2.5x)

input int EMA\_Period = 50; // EMA period

input double MaxDrawdownPercent = 2.5; // Maximum drawdown per day as a percentage of account balance

input int MaxTradesPerWeek = 2; // Maximum number of trades per week

// Global variables

double ConfluencesScore = 0;

string TimeFrames[] = {"W1", "D1", "H4"};

double dailyHigh = 0;

double dailyLow = 0;

datetime lastTradeTime = 0;

int weeklyTradeCount = 0; // Count the trades made in the current week

datetime lastTradeDate = 0; // Track the date of the last trade placed

double stopLoss = 0.0;

double takeProfit = 0.0;

// Define the valid symbols to trade

string ValidSymbols[] = {

"EURUSD", "GBPUSD", "USDJPY", "AUDUSD", "USDCAD", "NZDUSD", "CHFJPY", "GBPJPY", "AUDJPY",

"EURGBP", "EURJPY", "XAUUSD", "NAS100", "US30", "EURCHF", "GBPCHF", "AUDCHF","USDCHF","CADJPY",

"NZDCAD","AUDNZD","EURCAD","EURNZD","AUDCAD","EURAUD","NZDJPY","GBPNZD","GBPAUD","GBPCAD","GBPCHF"

};

//+------------------------------------------------------------------+

//| Initialization function |

//+------------------------------------------------------------------+

int OnInit()

{

ObjectCreate(0, "ConfluencesLabel", OBJ\_LABEL, 0, 0, 0);

ObjectSetInteger(0, "ConfluencesLabel", OBJPROP\_CORNER, CORNER\_RIGHT\_UPPER);

ObjectSetInteger(0, "ConfluencesLabel", OBJPROP\_XDISTANCE, 10);

ObjectSetInteger(0, "ConfluencesLabel", OBJPROP\_YDISTANCE, 10);

ObjectSetInteger(0, "ConfluencesLabel", OBJPROP\_COLOR, clrWhite);

ObjectSetInteger(0, "ConfluencesLabel", OBJPROP\_FONTSIZE, 10);

// Initialize daily high and low with the first candle of the day

int dayStartIndex = iBarShift(NULL, 0, iTime(NULL, 0, 0)); // Get the index of the first candle of the day

dailyHigh = High[dayStartIndex];

dailyLow = Low[dayStartIndex];

lastTradeDate = TimeDay(TimeCurrent()); // Initialize the last trade date

return(INIT\_SUCCEEDED);

}

//+------------------------------------------------------------------+

//| Deinitialization function |

//+------------------------------------------------------------------+

void OnDeinit(const int reason)

{

ObjectDelete(0, "ConfluencesLabel");

}

//+------------------------------------------------------------------+

//| Main function |

//+------------------------------------------------------------------+

void OnTick()

{

// Reset daily high and low at the start of a new day

if (TimeDay(TimeCurrent()) != TimeDay(lastTradeTime)) {

int dayStartIndex = iBarShift(NULL, 0, iTime(NULL, 0, 0)); // Get the index of the first candle of the day

dailyHigh = High[dayStartIndex];

dailyLow = Low[dayStartIndex];

lastTradeTime = TimeCurrent();

}

// Update daily high and low prices

if (High[0] > dailyHigh) dailyHigh = High[0];

if (Low[0] < dailyLow) dailyLow = Low[0];

ConfluencesScore = 0;

// Track the weekly trade count

if (TimeDayOfWeek(TimeCurrent()) == 1) // Check if it's a new week (Monday)

{

weeklyTradeCount = 0; // Reset trade count

}

// Identify areas of interest (support/resistance)

double areaOfInterestScore = IdentifyAreasOfInterest();

ConfluencesScore += areaOfInterestScore;

// Update the confluences score on the chart

UpdateConfluencesLabel();

// Check if we can place a trade (only 1-2 trades per week)

if (weeklyTradeCount < MaxTradesPerWeek && ConfluencesScore >= 70 && IsValidSymbol(Symbol()))

{

PlaceTrade();

}

}

//+------------------------------------------------------------------+

//| Update the confluences label on the chart |

//+------------------------------------------------------------------+

void UpdateConfluencesLabel()

{

string labelText = "Confluences Score: " + IntegerToString((int)ConfluencesScore);

ObjectSetString(0, "ConfluencesLabel", OBJPROP\_TEXT, labelText);

}

// Other functions remain unchanged...

//| Function to calculate and check areas of interest |

//+------------------------------------------------------------------+

double IdentifyAreasOfInterest()

{

double score = 0.0;

// Get support/resistance levels for weekly and daily timeframes

double supportLevelW1 = GetSupportResistance("W1", true); // Support level

double resistanceLevelW1 = GetSupportResistance("W1", false); // Resistance level

double supportLevelD1 = GetSupportResistance("D1", true);

double resistanceLevelD1 = GetSupportResistance("D1", false);

// Check if the price is near the area of interest

if (IsPriceNearLevel(supportLevelW1) || IsPriceNearLevel(resistanceLevelW1) ||

IsPriceNearLevel(supportLevelD1) || IsPriceNearLevel(resistanceLevelD1))

{

// Check for confluences (candlestick patterns, EMA, etc.)

if (CheckBodyTouches("W1") >= 3 || CheckBodyTouches("D1") >= 3)

score += 10.0; // 10% for areas of interest with 3 or more touches

if (CheckEMATouch("W1") || CheckEMATouch("D1"))

score += 5.0; // 5% for 50 EMA

if (CheckRejectionPattern("W1") || CheckRejectionPattern("D1"))

score += 10.0; // 10% for rejection patterns

if (CheckStructureRejection("W1") || CheckStructureRejection("D1"))

score += 10.0; // 10% for structure rejection

if (CheckRoundNumber("W1") || CheckRoundNumber("D1"))

score += 5.0; // 5% for round levels

if (CheckCandlestickPattern("W1") || CheckCandlestickPattern("D1"))

score += 10.0; // 10% for candlestick patterns

}

return score;

}

//+------------------------------------------------------------------+

//| Function to check if the price is near the support/resistance |

//+------------------------------------------------------------------+

bool IsPriceNearLevel(double level)

{

double tolerance = 50 \* Point; // 50 pips tolerance for support/resistance level

if (MathAbs(Ask - level) <= tolerance)

return true;

return false;

}

//+------------------------------------------------------------------+

//| Function to get support or resistance level (true for support) |

//+------------------------------------------------------------------+

double GetSupportResistance(string timeframe, bool isSupport)

{

int tf = StringToTimeframe(timeframe);

double level = 0;

// For simplicity, find the highest high for resistance and lowest low for support

int bars = iBars(Symbol(), tf);

if (isSupport)

{

level = Low[iLowest(Symbol(), tf, MODE\_LOW, bars, 0)]; // Find the lowest low (support)

}

else

{

level = High[iHighest(Symbol(), tf, MODE\_HIGH, bars, 0)]; // Find the highest high (resistance)

}

return level;

}

//+------------------------------------------------------------------+

//| Function to check for body touches on the chart |

//+------------------------------------------------------------------+

int CheckBodyTouches(string timeframe)

{

int touches = 0;

int tf = StringToTimeframe(timeframe);

// Get support and resistance levels for the given timeframe

double supportLevel = GetSupportResistance(timeframe, true);

double resistanceLevel = GetSupportResistance(timeframe, false);

int bars = iBars(Symbol(), tf);

// Loop through the past bars on the specified timeframe

for (int i = 0; i < bars; i++)

{

double open = iOpen(Symbol(), tf, i);

double close = iClose(Symbol(), tf, i);

double high = iHigh(Symbol(), tf, i);

double low = iLow(Symbol(), tf, i);

// Check if the body of the candle touches the support or resistance

if ((open >= supportLevel && close <= supportLevel) || (close >= supportLevel && open <= supportLevel))

{

touches++; // Body touches the support

}

if ((open >= resistanceLevel && close <= resistanceLevel) || (close >= resistanceLevel && open <= resistanceLevel))

{

touches++; // Body touches the resistance

}

}

return touches;

}

//+------------------------------------------------------------------+

//| Placeholder function to check for EMA touch |

//+------------------------------------------------------------------+

bool CheckEMATouch(string timeframe)

{

int tf = StringToTimeframe(timeframe);

double ema = iMA(Symbol(), tf, EMA\_Period, 0, MODE\_EMA, PRICE\_CLOSE, 0);

if (MathAbs(Ask - ema) <= (10 \* Point)) // 10 pips tolerance

return true;

return false;

}

//+------------------------------------------------------------------+

//| Check for a rejection pattern |

//+------------------------------------------------------------------+

bool CheckRejectionPattern(string timeframe)

{

int tf = StringToTimeframe(timeframe);

int bars = iBars(Symbol(), tf);

for (int i = 1; i <= 5; i++) // Check the last 5 bars

{

double open = iOpen(Symbol(), tf, i);

double close = iClose(Symbol(), tf, i);

double high = iHigh(Symbol(), tf, i);

double low = iLow(Symbol(), tf, i);

if (MathAbs(open - close) < (0.3 \* (high - low))) // Small body relative to the range

{

if (close > open && (high - close) > (2 \* (close - open))) // Bullish pin bar

return true;

if (close < open && (high - open) > (2 \* (close - open))) // Bearish pin bar

return true;

}

}

return false;

}

//+------------------------------------------------------------------+

//| Check for structure rejection |

//+------------------------------------------------------------------+

bool CheckStructureRejection(string timeframe)

{

int tf = StringToTimeframe(timeframe); // Convert timeframe to integer

int bars = iBars(Symbol(), tf); // Get the number of bars in the timeframe

if (bars < 3) return false; // Ensure there are enough bars for analysis

// Get the recent high/low points for structure analysis

double currentLow = iLow(Symbol(), tf, 1); // Current low

double previousLow = iLow(Symbol(), tf, 2); // Previous low

double secondPreviousLow = iLow(Symbol(), tf, 3); // Two bars ago low

double currentHigh = iHigh(Symbol(), tf, 1); // Current high

double previousHigh = iHigh(Symbol(), tf, 2); // Previous high

double secondPreviousHigh = iHigh(Symbol(), tf, 3); // Two bars ago high

// Check for buy rejection at a higher low (previous support holding)

if (currentLow > previousLow && previousLow > secondPreviousLow)

{

// Rejection at higher low for a buy signal

if (Ask > previousLow && Ask < currentLow) // Price is near the previous higher low

return true;

}

// Check for sell rejection at a lower high (previous resistance holding)

if (currentHigh < previousHigh && previousHigh < secondPreviousHigh)

{

// Rejection at lower high for a sell signal

if (Bid < previousHigh && Bid > currentHigh) // Price is near the previous lower high

return true;

}

return false; // No structure rejection detected

}

//+------------------------------------------------------------------+

//| Check if price is near a round number |

//+------------------------------------------------------------------+

bool CheckRoundNumber(string timeframe)

{

double price = NormalizeDouble(Ask, Digits);

if (MathAbs(price - MathRound(price / 0.01) \* 0.01) <= (10 \* Point)) // Adjust for round number tolerance

return true;

return false;

}

//+------------------------------------------------------------------+

//| Check for candlestick patterns |

//+------------------------------------------------------------------+

bool CheckCandlestickPattern(string timeframe)

{

int tf = StringToTimeframe(timeframe); // Convert the timeframe to an integer

int bars = iBars(Symbol(), tf); // Get the number of bars in the timeframe

if (bars < 3) return false; // Ensure there are enough bars for analysis

// Get OHLC data for the last 3 bars

double open1 = iOpen(Symbol(), tf, 1); // Current candle's open

double close1 = iClose(Symbol(), tf, 1); // Current candle's close

double high1 = iHigh(Symbol(), tf, 1); // Current candle's high

double low1 = iLow(Symbol(), tf, 1); // Current candle's low

double open2 = iOpen(Symbol(), tf, 2); // Previous candle's open

double close2 = iClose(Symbol(), tf, 2); // Previous candle's close

double high2 = iHigh(Symbol(), tf, 2); // Previous candle's high

double low2 = iLow(Symbol(), tf, 2); // Previous candle's low

double open3 = iOpen(Symbol(), tf, 3); // Candle before previous open

double close3 = iClose(Symbol(), tf, 3); // Candle before previous close

double high3 = iHigh(Symbol(), tf, 3); // Candle before previous high

double low3 = iLow(Symbol(), tf, 3); // Candle before previous low

// Pin Bar Detection

if (CheckPinBar(open1, close1, high1, low1)) return true;

// Engulfing Pattern Detection

if (CheckEngulfingPattern(open1, close1, open2, close2)) return true;

// Doji Detection

if (CheckDoji(open1, close1, high1, low1)) return true;

// Morning Star Detection

if (CheckMorningStar(open1, close1, open2, close2, open3, close3)) return true;

// Evening Star Detection

if (CheckEveningStar(open1, close1, open2, close2, open3, close3)) return true;

return false; // No patterns detected

}

bool CheckPinBar(double open, double close, double high, double low)

{

double bodySize = MathAbs(close - open);

double range = high - low;

double upperWick = high - MathMax(open, close);

double lowerWick = MathMin(open, close) - low;

// A Pin Bar has a small body and one long wick

if (bodySize < 0.3 \* range) // Body size < 30% of the range

{

if (upperWick > 2 \* bodySize || lowerWick > 2 \* bodySize) // Wick is at least 2x the body

return true;

}

return false;

}

bool CheckEngulfingPattern(double open1, double close1, double open2, double close2)

{

// Bullish Engulfing

if (close1 > open1 && open1 < close2 && close1 > open2) return true;

// Bearish Engulfing

if (close1 < open1 && open1 > close2 && close1 < open2) return true;

return false;

}

bool CheckDoji(double open, double close, double high, double low)

{

double bodySize = MathAbs(close - open);

double range = high - low;

// Doji has a very small body

if (bodySize <= 0.1 \* range) return true;

return false;

}

bool CheckMorningStar(double open1, double close1, double open2, double close2, double open3, double close3)

{

// Morning Star: Downtrend -> Small bearish/bullish candle -> Strong bullish candle

if (close3 < open3 && close2 < open2 && close1 > open1 && close1 > (open3 + close3) / 2)

return true;

return false;

}

bool CheckEveningStar(double open1, double close1, double open2, double close2, double open3, double close3)

{

// Evening Star: Uptrend -> Small bullish/bearish candle -> Strong bearish candle

if (close3 > open3 && close2 > open2 && close1 < open1 && close1 < (open3 + close3) / 2)

return true;

return false;

}

//+------------------------------------------------------------------+

//| Function to check if the symbol is valid for trading |

//+------------------------------------------------------------------+

bool IsValidSymbol(string symbol)

{

for (int i = 0; i < ArraySize(ValidSymbols); i++)

{

if (symbol == ValidSymbols[i])

{

return true; // The symbol is valid

}

}

return false; // The symbol is not valid

}

//+------------------------------------------------------------------+

//| Function to place a trade |

//+------------------------------------------------------------------+

void PlaceTrade() {

double lotSize = CalculateLotSize();

int tradeDirection = 0; // 1 for Buy, -1 for Sell

// Determine the previous structure points for SL

double previousSupport = GetSupportResistance("H4", true); // Support for Buy

double previousResistance = GetSupportResistance("H4", false); // Resistance for Sell

// Determine trade direction based on confluence score

if (ConfluencesScore > 70) { // Example threshold for buy

tradeDirection = 1; // Buy

stopLoss = previousSupport; // Use the global stopLoss variable

double riskPerPip = (AccountBalance() \* (RiskPercent / 100)) / lotSize;

double riskPips = MathAbs(Ask - stopLoss) / Point;

takeProfit = Ask + (riskPips \* TakeProfitFactor \* Point); // Assign takeProfit for Buy

} else if (ConfluencesScore < -70) { // Example threshold for sell

tradeDirection = -1; // Sell

stopLoss = previousResistance; // Use the global stopLoss variable

double riskPerPip = (AccountBalance() \* (RiskPercent / 100)) / lotSize;

double riskPips = MathAbs(Bid - stopLoss) / Point;

takeProfit = Bid - (riskPips \* TakeProfitFactor \* Point); // Assign takeProfit for Sell

} else {

Print("Conditions not met for trade. No trade executed.");

return; // Exit if conditions are not met

}

// Normalize SL and TP to match broker's precision

stopLoss = NormalizeDouble(stopLoss, Digits);

takeProfit = NormalizeDouble(takeProfit, Digits);

// Initialize ticket to an invalid value

int ticket = -1;

// Place the trade

if (tradeDirection == 1) { // Buy

ticket = OrderSend(Symbol(), OP\_BUY, lotSize, Ask, 3, stopLoss, takeProfit, "Buy Trade by EA", 0, 0, clrGreen);

} else if (tradeDirection == -1) { // Sell

ticket = OrderSend(Symbol(), OP\_SELL, lotSize, Bid, 3, stopLoss, takeProfit, "Sell Trade by EA", 0, 0, clrRed);

}

// Check if the order was successful

if (ticket > 0) {

weeklyTradeCount++;

lastTradeTime = TimeCurrent();

Print("Trade placed successfully. Ticket: ", ticket);

} else {

int errorCode = GetLastError();

Print("Failed to place trade. Error Code: ", errorCode);

// Handle specific errors

switch (errorCode) {

case 134: // ERR\_NO\_MONEY

Print("Not enough money to place the trade. Check account balance or reduce lot size.");

break;

case 131: // ERR\_INVALID\_TRADE\_VOLUME

Print("Invalid trade volume. Lot size: ", lotSize, ". Check broker's minimum and maximum lot size.");

break;

case 146: // ERR\_TRADE\_CONTEXT\_BUSY

Print("Trade context is busy. Retrying in 2 seconds...");

Sleep(2000);

PlaceTrade(); // Retry the trade

break;

default:

Print("Unhandled error: ", errorCode);

break;

}

}

}

//+------------------------------------------------------------------+

//| Function to calculate lot size |

//+------------------------------------------------------------------+

double CalculateLotSize()

{

double riskAmount = AccountBalance() \* (RiskPercent / 100);

double stopLossDistance = MathAbs(Ask - stopLoss) / Point;

double pipValue = MarketInfo(Symbol(), MODE\_TICKVALUE);

double lotSize = riskAmount / (stopLossDistance \* pipValue);

return NormalizeDouble(lotSize, 2); // Adjust for broker's lot size precision

}

//+------------------------------------------------------------------+

//| Function to convert string timeframe to integer |

//+------------------------------------------------------------------+

int StringToTimeframe(string timeframe)

{

if (timeframe == "M1") return PERIOD\_M1;

if (timeframe == "M5") return PERIOD\_M5;

if (timeframe == "M15") return PERIOD\_M15;

if (timeframe == "M30") return PERIOD\_M30;

if (timeframe == "H1") return PERIOD\_H1;

if (timeframe == "H4") return PERIOD\_H4;

if (timeframe == "D1") return PERIOD\_D1;

if (timeframe == "W1") return PERIOD\_W1;

if (timeframe == "MN1") return PERIOD\_MN1;

Print("Invalid timeframe: ", timeframe);

return 0; // Return 0 if invalid

}