```
//+------------------------------------------------------------------+
//|                                                 US30LagCons.mq5 |
//|        Prop Firm Quant Engine — SSRN LEAD-LAG CONSERVATIVE V1.50 |
//|        US30 (Dow Jones) — Lead-Lag Cross-Asset Relative Value    |
//+------------------------------------------------------------------+
#property copyright "Copyright 2026, Quantitative Prop Desk"
#property version   "1.50"
#property description "US30LagCons — SSRN Paper 1 Conservative Engine (1 Trade/Day Hard Cap, 0.50% Risk, Bulle
#property strict

#include <Trade/Trade.mqh>

#define MAGIC_US30LAGCONS_LONG   77717
#define MAGIC_US30LAGCONS_SHORT  77718

// --- PARAMETRI SSRN PAPER 1 CONSERVATIVE (LOCKED PASS #102 @ 0.50% RISK) ---
input group "=== 1. Asset Guida & Sessione Wall Street CET ==="
input string         InpLeadSymbol           = "USTEC";  // Asset Guida (Nasdaq 100)
input int            InpTradeStartHourCET    = 15;     // Ora Inizio CET (15:30 Wall Street Open)
input int            InpTradeStartMinCET     = 30;     // Minuto Inizio CET (30m)
input int            InpTradeEndHourCET      = 18;     // Fine Ingressi CET (18:30)
input int            InpEndHourCET           = 19;     // Hard Close Sessione CET (19:00)

input group "=== 2. Soglie Deviazione Lead-Lag & ATR (CONSERVATIVE) ==="
input double         InpLeadThresholdATR     = 1.20;   // Soglia Spinta USTEC (1.20 ATR Locked)
input double         InpLagMaxThresholdATR   = 0.50;   // Ritardo Max US30 per Ingresso (0.50 ATR Locked)
input int            InpADXPeriod            = 14;     // Periodo ADX (14)

input group "=== 3. Target, Stop ATR & Time Exit (CONSERVATIVE) ==="
input double         InpStopLossATR          = 0.80;   // Moltiplicatore ATR Stop Loss (0.80 ATR Locked)
input double         InpTakeProfitATR        = 4.50;   // Moltiplicatore ATR Take Profit (4.50 ATR Locked)
input int            InpMaxHoldingMins       = 90;     // Time Exit Max Minuti Holding (90 Minuti)

input group "=== 4. Rischio & Vincoli Prop Firm ==="
input double         InpRiskPct              = 0.50;   // Rischio per Trade (% Capitale — 0.50% Target)
input double         InpMaxLots              = 10.0;   // Limite Massimo Lotti
input double         InpMaxDailyLossPct      = 2.50;   // Circuit Breaker Max Loss Giornaliera (%)
input double         InpMaxSpreadPoints      = 300.0;  // Max Spread Consentito

input group "=== 5. Vincolo Posizione Singola & 1 Trade/Giorno (Hard Safety Cap) ==="
input int            InpMaxOpenPositions     = 1;      // Max Posizioni Aperte Contemporaneamente (Tassativo:
input bool           InpMaxOneTradePerDay    = true;   // Limite Tassativo Max 1 Trade al Giorno (TASSATIVO: t

// --- Global Variables ---
CTrade   trade;
int      atr_us30_handle;
int      atr_ustec_handle;

datetime last_bar_time = 0;
double   daily_start_equity = 0.0;
int      daily_day = 0;
bool     traded_today = false;

double NormalizeLot(double lot)
{
   double min_lot  = SymbolInfoDouble(_Symbol, SYMBOL_VOLUME_MIN);
   double max_lot  = SymbolInfoDouble(_Symbol, SYMBOL_VOLUME_MAX);
   double step_lot = SymbolInfoDouble(_Symbol, SYMBOL_VOLUME_STEP);

   if(step_lot <= 0) step_lot = 0.01;
   if(min_lot <= 0)  min_lot  = 0.01;
```

```
   if(max_lot <= 0)  max_lot  = 1000.0;

   double steps = MathFloor(lot / step_lot + 0.000001);
   double normalized = steps * step_lot;

   int digits = 0;
   if(step_lot < 1.0)
   {
      double temp = step_lot;
      while(temp < 0.999999)
      {
         temp *= 10.0;
         digits++;
         if(digits >= 8) break;
      }
   }

   normalized = NormalizeDouble(normalized, digits);

   if(normalized < min_lot) normalized = min_lot;
   if(normalized > max_lot) normalized = max_lot;

   return normalized;
}

int OnInit()
{
   trade.SetMarginMode();
   trade.SetTypeFillingBySymbol(_Symbol);
   trade.SetTypeFilling(ORDER_FILLING_IOC);

   atr_us30_handle  = iATR(_Symbol, PERIOD_M15, 14);
   atr_ustec_handle = iATR(InpLeadSymbol, PERIOD_M15, 14);

   if(atr_us30_handle == INVALID_HANDLE || atr_ustec_handle == INVALID_HANDLE) {
      Print("[US30LagCons] Errore Inizializzazione Indicatori! Verifica presenza del simbolo USTEC.");
      return(INIT_FAILED);
   }

   Print("[US30LagCons] SSRN Paper 1 Conservative Engine (1 Trade/Day Cap @ 0.50% Risk) Inizializzato su ", _S
   return(INIT_SUCCEEDED);
}

void OnDeinit(const int reason)
{
   IndicatorRelease(atr_us30_handle);
   IndicatorRelease(atr_ustec_handle);
}

void CloseAllPositions(string reason)
{
   for(int i = PositionsTotal() - 1; i >= 0; i--) {
      ulong ticket = PositionGetTicket(i);
      if(ticket > 0 && PositionGetString(POSITION_SYMBOL) == _Symbol) {
         long magic = PositionGetInteger(POSITION_MAGIC);
         if(magic == MAGIC_US30LAGCONS_LONG || magic == MAGIC_US30LAGCONS_SHORT) {
            trade.PositionClose(ticket);
            PrintFormat("[US30LagCons] Chiusura Posizione #%d: %s", ticket, reason);
         }
      }
   }
}
```

```
bool CheckPropFirmRiskLimits()
{
   datetime now = TimeCurrent();
   MqlDateTime dt;
   TimeToStruct(now, dt);

   if(dt.day != daily_day) {
      daily_day = dt.day;
      daily_start_equity = AccountInfoDouble(ACCOUNT_EQUITY);
      traded_today = false;
   }

   if(daily_start_equity > 0) {
      double current_equity = AccountInfoDouble(ACCOUNT_EQUITY);
      double loss_pct = ((daily_start_equity - current_equity) / daily_start_equity) * 100.0;
      if(loss_pct >= InpMaxDailyLossPct) {
         CloseAllPositions("Circuit Breaker Prop Firm (Daily Loss Limit)");
         return true;
      }
   }

   int end_server = InpEndHourCET + 1; // GMT+3
   if(dt.hour >= end_server) {
      CloseAllPositions("Hard Close Sessione 19:00 CET");
      return true;
   }

   return false;
}

void CheckTimeExit()
{
   if(InpMaxHoldingMins <= 0) return;

   datetime now = TimeCurrent();

   for(int i = PositionsTotal() - 1; i >= 0; i--) {
      ulong ticket = PositionGetTicket(i);
      if(ticket > 0 && PositionGetString(POSITION_SYMBOL) == _Symbol) {
         long magic = PositionGetInteger(POSITION_MAGIC);
         if(magic == MAGIC_US30LAGCONS_LONG || magic == MAGIC_US30LAGCONS_SHORT) {
            datetime open_time = (datetime)PositionGetInteger(POSITION_TIME);
            int duration_mins = (int)((now - open_time) / 60);
            if(duration_mins >= InpMaxHoldingMins) {
               trade.PositionClose(ticket);
               PrintFormat("[US30LagCons] Time Exit Scattato dopo %d minuti! Ticket: %d", duration_mins, ticke
            }
         }
      }
   }
}

int TotalOpenPositions()
{
   int count = 0;
   for(int i = PositionsTotal() - 1; i >= 0; i--) {
      ulong ticket = PositionGetTicket(i);
      if(ticket > 0 && PositionGetString(POSITION_SYMBOL) == _Symbol) {
         long magic = PositionGetInteger(POSITION_MAGIC);
         if(magic == MAGIC_US30LAGCONS_LONG || magic == MAGIC_US30LAGCONS_SHORT) {
            count++;
         }
      }
```

```
   }
   return count;
}

void ExecuteOrder(ENUM_ORDER_TYPE type, double price, double atr, ulong magic, string comment)
{
   if(TotalOpenPositions() >= InpMaxOpenPositions || (InpMaxOneTradePerDay && traded_today)) {
      return;
   }

   trade.SetExpertMagicNumber(magic);

   double balance = AccountInfoDouble(ACCOUNT_BALANCE);
   double risk_amount = balance * (InpRiskPct / 100.0);

   double sl_dist = atr * InpStopLossATR;
   double tp_dist = atr * InpTakeProfitATR;

   double sl = (type == ORDER_TYPE_BUY) ? NormalizeDouble(price - sl_dist, _Digits) : NormalizeDouble(price +
   double tp = (type == ORDER_TYPE_BUY) ? NormalizeDouble(price + tp_dist, _Digits) : NormalizeDouble(price -

   double tick_value = SymbolInfoDouble(_Symbol, SYMBOL_TRADE_TICK_VALUE);
   double tick_size  = SymbolInfoDouble(_Symbol, SYMBOL_TRADE_TICK_SIZE);

   double loss_per_lot = (sl_dist / tick_size) * tick_value;
   double raw_lot = (loss_per_lot > 0) ? NormalizeDouble(risk_amount / loss_per_lot, 2) : 0.10;

   if(InpMaxLots > 0 && raw_lot > InpMaxLots) raw_lot = InpMaxLots;

   double lot_size = NormalizeLot(raw_lot);

   bool success = false;
   if(type == ORDER_TYPE_BUY) {
      success = trade.Buy(lot_size, _Symbol, price, sl, tp, comment);
   } else {
      success = trade.Sell(lot_size, _Symbol, price, sl, tp, comment);
   }

   if(success) {
      traded_today = true;
      PrintFormat("[US30LagCons] Ordine Singolo Eseguito! Lot: %.2f | Price: %.2f | SL: %.2f | TP: %.2f", lot_
   }
}

void OnTick()
{
   if(CheckPropFirmRiskLimits()) return;
   CheckTimeExit();

   datetime current_bar = iTime(_Symbol, PERIOD_M15, 0);
   if(current_bar == last_bar_time) return;
   last_bar_time = current_bar;

   if(TotalOpenPositions() >= InpMaxOpenPositions || (InpMaxOneTradePerDay && traded_today)) return;

   MqlDateTime dt;
   TimeToStruct(TimeCurrent(), dt);

   // Entry Window: 15:30 CET to 18:30 CET (16:30 GMT+3 to 19:30 GMT+3)
   int server_trade_start_hour = InpTradeStartHourCET + 1; // 16:30 GMT+3
   int server_trade_end_hour   = InpTradeEndHourCET + 1;   // 19:30 GMT+3

   if(dt.hour < server_trade_start_hour || (dt.hour == server_trade_start_hour && dt.min < InpTradeStartMinCET
```

```
   if(dt.hour >= server_trade_end_hour) return;

   double ask = SymbolInfoDouble(_Symbol, SYMBOL_ASK);
   double bid = SymbolInfoDouble(_Symbol, SYMBOL_BID);
   double spread = ask - bid;
   if(spread > InpMaxSpreadPoints * _Point) return;

   // 1. Calculate USTEC M15 Return vs USTEC ATR
   MqlRates ustec_rates[];
   ArraySetAsSeries(ustec_rates, true);
   if(CopyRates(InpLeadSymbol, PERIOD_M15, 1, 2, ustec_rates) < 2) return;

   double ustec_ret = ustec_rates[0].close - ustec_rates[1].close;

   double ustec_atr_val[];
   ArraySetAsSeries(ustec_atr_val, true);
   if(CopyBuffer(atr_ustec_handle, 0, 1, 1, ustec_atr_val) <= 0) return;
   double ustec_atr = ustec_atr_val[0];

   // 2. Calculate US30 M15 Return vs US30 ATR
   MqlRates us30_rates[];
   ArraySetAsSeries(us30_rates, true);
   if(CopyRates(_Symbol, PERIOD_M15, 1, 2, us30_rates) < 2) return;

   double us30_ret = us30_rates[0].close - us30_rates[1].close;

   double us30_atr_val[];
   ArraySetAsSeries(us30_atr_val, true);
   if(CopyBuffer(atr_us30_handle, 0, 1, 1, us30_atr_val) <= 0) return;
   double us30_atr = us30_atr_val[0];

   // SSRN LEAD-LAG ARBITRAGE SIGNAL:
   if(ustec_ret > InpLeadThresholdATR * ustec_atr && us30_ret < InpLagMaxThresholdATR * us30_atr) {
      ExecuteOrder(ORDER_TYPE_BUY, ask, us30_atr, MAGIC_US30LAGCONS_LONG, "SSRN Lead-Lag Buy US30");
   }
   else if(ustec_ret < -InpLeadThresholdATR * ustec_atr && us30_ret > -InpLagMaxThresholdATR * us30_atr) {
      ExecuteOrder(ORDER_TYPE_SELL, bid, us30_atr, MAGIC_US30LAGCONS_SHORT, "SSRN Lead-Lag Sell US30");
   }
}
```