```
//+------------------------------------------------------------------+
//|  TS Auto Risk Panel EA (FTMO MT5)                                |
//|  Single panel: [SELL] [Risk] [BUY] + [OFF]                       |
//|  On button: FIRST trade is immediate (manual trigger)            |
//|  Then: auto entries on TS signal (bar close) in current chart TF |
//|  SL: wick +/- monetary buffer (included in Risk)                 |
//|  Limits: MaxEntries, StopAfterSL                                 |
//+------------------------------------------------------------------+
#property strict

#include <Trade/Trade.mqh>
CTrade trade;

//---------------------- Inputs ----------------------//
input long   InpMagicNumber      = 240205; // Magic number
input double InpDefaultRiskEUR   = 50.0;   // Default Risk (EUR) shown in panel
input double InpBufferEUR        = 5.0;    // Buffer in EUR (included within Risk)
input int    InpMaxEntries       = 5;      // Max concurrent entries (EA+symbol+direction)
input int    InpStopAfterSL      = 3;      // Stop trading after N SL hits
input int    InpSlippagePoints   = 20;     // Deviation in points
input bool   InpAllowMultiplePos = true;   // Hedging: allow multiple positions

//---------------------- Mode ----------------------//
enum TradeMode
{
   MODE_OFF = 0,
   MODE_BUY_ONLY = 1,
   MODE_SELL_ONLY = 2
};
TradeMode g_mode = MODE_OFF;

int g_sl_hits = 0;

datetime g_last_bar_time = 0;
datetime g_last_signal_time = 0;

//---------------------- UI names ----------------------//
string UI_PREFIX;
string OBJ_BG, OBJ_SELL, OBJ_BUY, OBJ_OFF, OBJ_RISK_EDIT, OBJ_RISK_LBL,
OBJ_STATUS;

//---------------------- Helpers ----------------------//
double GetSymbolDouble(const string sym, const ENUM_SYMBOL_INFO_DOUBLE prop)
{
   double v=0.0;
   if(!SymbolInfoDouble(sym, prop, v)) return 0.0;
   return v;
}

double GetSymbolPoint(const string sym)
{
   double p = GetSymbolDouble(sym, SYMBOL_POINT);
```

```
  if(p <= 0) p = 0.00001;
  return p;
}

// k = money per 1.0 price move per 1 lot in deposit currency
bool MoneyPerPriceUnit_1Lot(const string sym, double &k_out)
{
  double tick_size  = GetSymbolDouble(sym, SYMBOL_TRADE_TICK_SIZE);
  double tick_value = GetSymbolDouble(sym, SYMBOL_TRADE_TICK_VALUE);

  if(tick_size <= 0.0) tick_size = GetSymbolDouble(sym, SYMBOL_POINT);
  if(tick_size <= 0.0) tick_size = GetSymbolPoint(sym);

  if(tick_value <= 0.0)
  {
    double bid=0, ask=0;
    if(!SymbolInfoDouble(sym, SYMBOL_BID, bid) || !SymbolInfoDouble(sym, SYMBOL_ASK,
ask)) return false;
    double mid = (bid+ask)*0.5;
    double profit=0.0;

    if(!OrderCalcProfit(ORDER_TYPE_BUY, sym, 1.0, mid, mid + tick_size, profit)) return false;
    tick_value = MathAbs(profit);
    if(tick_value <= 0.0) return false;
  }

  k_out = tick_value / tick_size;
  return (k_out > 0.0);
}

double NormalizeVolumeDown(const string sym, const double vol)
{
  double vmin = GetSymbolDouble(sym, SYMBOL_VOLUME_MIN);
  double vmax = GetSymbolDouble(sym, SYMBOL_VOLUME_MAX);
  double step = GetSymbolDouble(sym, SYMBOL_VOLUME_STEP);

  if(step <= 0) step = 0.01;
  if(vmin <= 0) vmin = step;
  if(vmax <= 0) vmax = 100.0;

  double v = vol;
  v = MathFloor(v / step) * step;

  if(v > vmax) v = vmax;
  if(v < vmin) return 0.0;

  int digits = 0;
  if(step < 1.0) digits = (int)MathRound(-MathLog10(step));
  return NormalizeDouble(v, digits);
}

double NormalizePriceToTick(const string sym, const double price)
```

```
{
  int digits = (int)SymbolInfoInteger(sym, SYMBOL_DIGITS);

  double tick_size = GetSymbolDouble(sym, SYMBOL_TRADE_TICK_SIZE);
  if(tick_size <= 0.0) tick_size = GetSymbolDouble(sym, SYMBOL_POINT);
  if(tick_size <= 0.0) tick_size = GetSymbolPoint(sym);

  double p = MathRound(price / tick_size) * tick_size;
  return NormalizeDouble(p, digits);
}

double ReadRiskFromUI()
{
  string s = ObjectGetString(0, OBJ_RISK_EDIT, OBJPROP_TEXT);
  StringTrimLeft(s);
  StringTrimRight(s);

  double r = StringToDouble(s);
  if(r <= 0) r = InpDefaultRiskEUR;
  return r;
}

string ModeToText()
{
  if(g_mode == MODE_BUY_ONLY)  return "MODE: BUY ONLY";
  if(g_mode == MODE_SELL_ONLY) return "MODE: SELL ONLY";
  return "MODE: OFF";
}

void SetStatus(const string msg)
{
  ObjectSetString(0, OBJ_STATUS, OBJPROP_TEXT, msg);
}

int CountOpenPositions(const string sym, const bool isBuy)
{
  int count = 0;
  for(int i=0; i<PositionsTotal(); i++)
  {
    ulong ticket = PositionGetTicket(i);
    if(ticket == 0) continue;
    if(!PositionSelectByTicket(ticket)) continue;

    if(PositionGetString(POSITION_SYMBOL) != sym) continue;
    if((long)PositionGetInteger(POSITION_MAGIC) != InpMagicNumber) continue;

    long type = PositionGetInteger(POSITION_TYPE);
    if(isBuy && type == POSITION_TYPE_BUY) count++;
    if(!isBuy && type == POSITION_TYPE_SELL) count++;
  }
  return count;
}
```

```
bool TradingAllowedNow()
{
  if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED)) return false;
  if(!MQLInfoInteger(MQL_TRADE_ALLOWED)) return false;
  if(AccountInfoInteger(ACCOUNT_TRADE_ALLOWED) == 0) return false;
  return true;
}

//---------------------- TS detection (bar closed) ----------------------//
bool IsTSBull(const string sym, const ENUM_TIMEFRAMES tf)
{
  if(Bars(sym, tf) < 3) return false;
  double lowB   = iLow(sym, tf, 1);
  double lowA   = iLow(sym, tf, 2);
  double openB  = iOpen(sym, tf, 1);
  double closeB = iClose(sym, tf, 1);

  return (lowB < lowA && closeB > openB);
}

bool IsTSBear(const string sym, const ENUM_TIMEFRAMES tf)
{
  if(Bars(sym, tf) < 3) return false;
  double highB  = iHigh(sym, tf, 1);
  double highA  = iHigh(sym, tf, 2);
  double openB  = iOpen(sym, tf, 1);
  double closeB = iClose(sym, tf, 1);

  return (highB > highA && closeB < openB);
}

//---------------------- Trade placement (core) ----------------------//
// usePrevCandleWick=true  => SL anchored to previous candle wick (shift 1)
// usePrevCandleWick=false => SL anchored to TS candle wick (shift 1 as signal bar - handled
outside)
bool PlaceTradeWithWickShift(const bool isBuy, const int wickShift, const string tag)
{
  const string sym = _Symbol;
  const ENUM_TIMEFRAMES tf = (ENUM_TIMEFRAMES)_Period;

  if(!TradingAllowedNow())
  {
    SetStatus("Trading not allowed now.");
    return false;
  }

  if(g_sl_hits >= InpStopAfterSL)
  {
    g_mode = MODE_OFF;
    SetStatus("Stopped: SL hits reached. MODE OFF.");
    return false;
```

```
   }

   // Max entries per direction
   int openCount = CountOpenPositions(sym, isBuy);
   if(openCount >= InpMaxEntries)
   {
      SetStatus("Max entries reached (" + IntegerToString(InpMaxEntries) + ").");
      return false;
   }

   // Hedging option
   if(!InpAllowMultiplePos)
   {
      if(PositionSelect(sym))
      {
         SetStatus("Position exists on symbol (blocked).");
         return false;
      }
   }

   if(Bars(sym, tf) < (wickShift + 2))
   {
      SetStatus("Not enough bars for wick shift.");
      return false;
   }

   double bid=0, ask=0;
   if(!SymbolInfoDouble(sym, SYMBOL_BID, bid) || !SymbolInfoDouble(sym, SYMBOL_ASK,
ask))
   {
      SetStatus("Cannot read BID/ASK.");
      return false;
   }
   double entry = isBuy ? ask : bid;

   // Wick from specified shift
   double wick = isBuy ? iLow(sym, tf, wickShift) : iHigh(sym, tf, wickShift);

   double d0 = isBuy ? (entry - wick) : (wick - entry);
   if(d0 <= 0)
   {
      SetStatus("Distance to wick <= 0.");
      return false;
   }

   double RiskEUR   = ReadRiskFromUI();
   double BufferEUR = InpBufferEUR;

   if(RiskEUR <= BufferEUR)
   {
      SetStatus("Risk must be > Buffer.");
      return false;
```

```
}

double k=0.0;
if(!MoneyPerPriceUnit_1Lot(sym, k))
{
  SetStatus("Cannot compute tick value/size.");
  return false;
}

double lot_raw = (RiskEUR - BufferEUR) / (k * d0);
if(lot_raw <= 0)
{
  SetStatus("Lot <= 0.");
  return false;
}

double lot = NormalizeVolumeDown(sym, lot_raw);
if(lot <= 0.0)
{
  SetStatus("Lot below minimum. Increase risk.");
  return false;
}

double db = BufferEUR / (lot * k);

double sl = isBuy ? (wick - db) : (wick + db);
sl = NormalizePriceToTick(sym, sl);

if(isBuy && sl >= entry)
{
  SetStatus("Invalid SL (BUY).");
  return false;
}
if(!isBuy && sl <= entry)
{
  SetStatus("Invalid SL (SELL).");
  return false;
}

double loss=0.0;
ENUM_ORDER_TYPE type = isBuy ? ORDER_TYPE_BUY : ORDER_TYPE_SELL;
if(!OrderCalcProfit(type, sym, lot, entry, sl, loss))
{
  SetStatus("Cannot calc final risk.");
  return false;
}
double risk_est = MathAbs(loss);

// If above target due to rounding, reduce one step
double step = GetSymbolDouble(sym, SYMBOL_VOLUME_STEP);
if(step <= 0) step = 0.01;
```

```
  if(risk_est > RiskEUR * 1.002)
  {
    double lot2 = NormalizeVolumeDown(sym, lot - step);
    if(lot2 > 0.0)
    {
      lot = lot2;
      db  = BufferEUR / (lot * k);
      sl  = isBuy ? (wick - db) : (wick + db);
      sl  = NormalizePriceToTick(sym, sl);

      if(!OrderCalcProfit(type, sym, lot, entry, sl, loss))
      {
        SetStatus("Cannot recalc risk.");
        return false;
      }
      risk_est = MathAbs(loss);
    }
  }

  trade.SetExpertMagicNumber(InpMagicNumber);
  trade.SetDeviationInPoints(InpSlippagePoints);

  bool ok=false;
  if(isBuy)
    ok = trade.Buy(lot, sym, entry, sl, 0.0, tag);
  else
    ok = trade.Sell(lot, sym, entry, sl, 0.0, tag);

  if(!ok)
  {
    SetStatus("Rejected: " + IntegerToString(trade.ResultRetcode()) + " | " +
trade.ResultRetcodeDescription());
    return false;
  }

  string side = isBuy ? "BUY" : "SELL";
  SetStatus(ModeToText() + " | " + side + " placed | lot=" + DoubleToString(lot,2) +
         " | Risk~" + DoubleToString(risk_est,2) + " EUR | SLhits=" + IntegerToString(g_sl_hits));
  return true;
}

// Manual-triggered first trade: wickShift=1 => previous candle
bool PlaceFirstManualTrade(const bool isBuy)
{
  return PlaceTradeWithWickShift(isBuy, 1, isBuy ? "FIRST_MANUAL_BUY" :
"FIRST_MANUAL_SELL");
}

// Auto trade on TS signal: use TS candle wick (bar[1]) => wickShift=1 (signal bar is bar[1] on new
bar)
bool PlaceAutoTradeOnSignal(const bool isBuy)
{
  return PlaceTradeWithWickShift(isBuy, 1, isBuy ? "TS_AUTO_BUY" : "TS_AUTO_SELL");
}

//---------------------- UI ----------------------//
void CreateUI()
{
  UI_PREFIX     = "TS_AutoPanel_" + IntegerToString((int)ChartID()) + "_";
  OBJ_BG        = UI_PREFIX + "BG";
  OBJ_SELL      = UI_PREFIX + "SELL";
  OBJ_BUY       = UI_PREFIX + "BUY";
  OBJ_OFF       = UI_PREFIX + "OFF";
  OBJ_RISK_EDIT = UI_PREFIX + "RiskEdit";
  OBJ_RISK_LBL  = UI_PREFIX + "RiskLbl";
  OBJ_STATUS    = UI_PREFIX + "Status";

  int x=18, y=20;
  int panel_w=320;
  int panel_h=64;

  int top_h=28;
  int gap=6;

  int btn_w=78;
  int edit_w=74;
  int row_y=y+8;

  ObjectCreate(0, OBJ_BG, OBJ_RECTANGLE_LABEL, 0, 0, 0);
  ObjectSetInteger(0, OBJ_BG, OBJPROP_CORNER, CORNER_LEFT_UPPER);
  ObjectSetInteger(0, OBJ_BG, OBJPROP_XDISTANCE, x);
  ObjectSetInteger(0, OBJ_BG, OBJPROP_YDISTANCE, y);
  ObjectSetInteger(0, OBJ_BG, OBJPROP_XSIZE, panel_w);
  ObjectSetInteger(0, OBJ_BG, OBJPROP_YSIZE, panel_h);
  ObjectSetInteger(0, OBJ_BG, OBJPROP_BACK, false);
  ObjectSetInteger(0, OBJ_BG, OBJPROP_COLOR, clrBlack);
  ObjectSetInteger(0, OBJ_BG, OBJPROP_BGCOLOR, clrDimGray);
  ObjectSetInteger(0, OBJ_BG, OBJPROP_BORDER_TYPE, BORDER_FLAT);

  ObjectCreate(0, OBJ_SELL, OBJ_BUTTON, 0, 0, 0);
  ObjectSetInteger(0, OBJ_SELL, OBJPROP_CORNER, CORNER_LEFT_UPPER);
  ObjectSetInteger(0, OBJ_SELL, OBJPROP_XDISTANCE, x+8);
  ObjectSetInteger(0, OBJ_SELL, OBJPROP_YDISTANCE, row_y);
  ObjectSetInteger(0, OBJ_SELL, OBJPROP_XSIZE, btn_w);
  ObjectSetInteger(0, OBJ_SELL, OBJPROP_YSIZE, top_h);
  ObjectSetString (0, OBJ_SELL, OBJPROP_TEXT, "SELL");
  ObjectSetInteger(0, OBJ_SELL, OBJPROP_COLOR, clrWhite);
  ObjectSetInteger(0, OBJ_SELL, OBJPROP_BGCOLOR, clrFireBrick);

  int edit_x = x+8+btn_w+gap;
  ObjectCreate(0, OBJ_RISK_EDIT, OBJ_EDIT, 0, 0, 0);
  ObjectSetInteger(0, OBJ_RISK_EDIT, OBJPROP_CORNER, CORNER_LEFT_UPPER);
  ObjectSetInteger(0, OBJ_RISK_EDIT, OBJPROP_XDISTANCE, edit_x);
  ObjectSetInteger(0, OBJ_RISK_EDIT, OBJPROP_YDISTANCE, row_y);
```

```
  ObjectSetInteger(0, OBJ_RISK_EDIT, OBJPROP_XSIZE, edit_w);
  ObjectSetInteger(0, OBJ_RISK_EDIT, OBJPROP_YSIZE, top_h);
  ObjectSetString (0, OBJ_RISK_EDIT, OBJPROP_TEXT, DoubleToString(InpDefaultRiskEUR,
0));
  ObjectSetInteger(0, OBJ_RISK_EDIT, OBJPROP_COLOR, clrBlack);
  ObjectSetInteger(0, OBJ_RISK_EDIT, OBJPROP_BGCOLOR, clrWhite);

  ObjectCreate(0, OBJ_RISK_LBL, OBJ_LABEL, 0, 0, 0);
  ObjectSetInteger(0, OBJ_RISK_LBL, OBJPROP_CORNER, CORNER_LEFT_UPPER);
  ObjectSetInteger(0, OBJ_RISK_LBL, OBJPROP_XDISTANCE, edit_x + edit_w + 4);
  ObjectSetInteger(0, OBJ_RISK_LBL, OBJPROP_YDISTANCE, row_y + 6);
  ObjectSetInteger(0, OBJ_RISK_LBL, OBJPROP_FONTSIZE, 10);
  ObjectSetInteger(0, OBJ_RISK_LBL, OBJPROP_COLOR, clrWhite);
  ObjectSetString (0, OBJ_RISK_LBL, OBJPROP_TEXT, "EUR");

  int buy_x = edit_x + edit_w + 32;
  ObjectCreate(0, OBJ_BUY, OBJ_BUTTON, 0, 0, 0);
  ObjectSetInteger(0, OBJ_BUY, OBJPROP_CORNER, CORNER_LEFT_UPPER);
  ObjectSetInteger(0, OBJ_BUY, OBJPROP_XDISTANCE, buy_x);
  ObjectSetInteger(0, OBJ_BUY, OBJPROP_YDISTANCE, row_y);
  ObjectSetInteger(0, OBJ_BUY, OBJPROP_XSIZE, btn_w);
  ObjectSetInteger(0, OBJ_BUY, OBJPROP_YSIZE, top_h);
  ObjectSetString (0, OBJ_BUY, OBJPROP_TEXT, "BUY");
  ObjectSetInteger(0, OBJ_BUY, OBJPROP_COLOR, clrWhite);
  ObjectSetInteger(0, OBJ_BUY, OBJPROP_BGCOLOR, clrDodgerBlue);

  int off_x = buy_x + btn_w + 8;
  ObjectCreate(0, OBJ_OFF, OBJ_BUTTON, 0, 0, 0);
  ObjectSetInteger(0, OBJ_OFF, OBJPROP_CORNER, CORNER_LEFT_UPPER);
  ObjectSetInteger(0, OBJ_OFF, OBJPROP_XDISTANCE, off_x);
  ObjectSetInteger(0, OBJ_OFF, OBJPROP_YDISTANCE, row_y);
  ObjectSetInteger(0, OBJ_OFF, OBJPROP_XSIZE, 44);
  ObjectSetInteger(0, OBJ_OFF, OBJPROP_YSIZE, top_h);
  ObjectSetString (0, OBJ_OFF, OBJPROP_TEXT, "OFF");
  ObjectSetInteger(0, OBJ_OFF, OBJPROP_COLOR, clrWhite);
  ObjectSetInteger(0, OBJ_OFF, OBJPROP_BGCOLOR, clrGray);

  ObjectCreate(0, OBJ_STATUS, OBJ_LABEL, 0, 0, 0);
  ObjectSetInteger(0, OBJ_STATUS, OBJPROP_CORNER, CORNER_LEFT_UPPER);
  ObjectSetInteger(0, OBJ_STATUS, OBJPROP_XDISTANCE, x+10);
  ObjectSetInteger(0, OBJ_STATUS, OBJPROP_YDISTANCE, y+40);
  ObjectSetInteger(0, OBJ_STATUS, OBJPROP_FONTSIZE, 9);
  ObjectSetInteger(0, OBJ_STATUS, OBJPROP_COLOR, clrGainsboro);

  SetStatus(ModeToText() + " | Buffer " + DoubleToString(InpBufferEUR,2) +
            " | MaxEntries " + IntegerToString(InpMaxEntries) +
            " | StopAfterSL " + IntegerToString(InpStopAfterSL) +
            " | SLhits " + IntegerToString(g_sl_hits));

  ChartRedraw();
}
```

```
void DeleteUI()
{
  int total = ObjectsTotal(0, 0, -1);
  for(int i=total-1; i>=0; i--)
  {
    string name = ObjectName(0, i, 0, -1);
    if(StringFind(name, UI_PREFIX) == 0)
      ObjectDelete(0, name);
  }
}

//---------------------- Lifecycle ----------------------//
int OnInit()
{
  g_last_bar_time = iTime(_Symbol, (ENUM_TIMEFRAMES)_Period, 0);
  g_last_signal_time = 0;
  g_sl_hits = 0;

  CreateUI();
  trade.SetExpertMagicNumber(InpMagicNumber);
  return(INIT_SUCCEEDED);
}

void OnDeinit(const int reason)
{
  DeleteUI();
}

// Count SL hits using trade transactions (best for FTMO hedging)
void OnTradeTransaction(const MqlTradeTransaction& trans,
                        const MqlTradeRequest& request,
                        const MqlTradeResult& result)
{
  if(trans.type != TRADE_TRANSACTION_DEAL_ADD) return;

  ulong deal = trans.deal;
  if(deal == 0) return;

  string sym = HistoryDealGetString(deal, DEAL_SYMBOL);
  if(sym != _Symbol) return;

  long magic = (long)HistoryDealGetInteger(deal, DEAL_MAGIC);
  if(magic != InpMagicNumber) return;

  long entry = (long)HistoryDealGetInteger(deal, DEAL_ENTRY);
  if(entry != DEAL_ENTRY_OUT && entry != DEAL_ENTRY_OUT_BY) return;

  long reason = (long)HistoryDealGetInteger(deal, DEAL_REASON);
  if(reason == DEAL_REASON_SL)
  {
    g_sl_hits++;
    if(g_sl_hits >= InpStopAfterSL)
```

```
      {
         g_mode = MODE_OFF;
         SetStatus("SL hit #" + IntegerToString(g_sl_hits) + " -> MODE OFF (stopped).");
      }
      else
      {
         SetStatus(ModeToText() + " | SL hit #" + IntegerToString(g_sl_hits));
      }
   }
}

//---------------------- Main loop: act on new bar ----------------------//
void OnTick()
{
   const string sym = _Symbol;
   const ENUM_TIMEFRAMES tf = (ENUM_TIMEFRAMES)_Period;

   datetime t0 = iTime(sym, tf, 0);
   if(t0 == 0) return;

   if(t0 == g_last_bar_time) return; // not a new bar
   g_last_bar_time = t0;

   if(g_mode == MODE_OFF) return;
   if(g_sl_hits >= InpStopAfterSL)
   {
      g_mode = MODE_OFF;
      SetStatus("Stopped: SL hits reached. MODE OFF.");
      return;
   }

   datetime signal_time = iTime(sym, tf, 1);
   if(signal_time == 0) return;
   if(signal_time == g_last_signal_time) return;

   if(g_mode == MODE_BUY_ONLY)
   {
      if(IsTSBull(sym, tf))
      {
         g_last_signal_time = signal_time;
         PlaceAutoTradeOnSignal(true);
      }
      else
      {
         SetStatus(ModeToText() + " | No TS bull | SLhits=" + IntegerToString(g_sl_hits));
      }
   }
   else if(g_mode == MODE_SELL_ONLY)
   {
      if(IsTSBear(sym, tf))
      {
         g_last_signal_time = signal_time;
      PlaceAutoTradeOnSignal(false);
    }
    else
    {
      SetStatus(ModeToText() + " | No TS bear | SLhits=" + IntegerToString(g_sl_hits));
    }
  }
}

//---------------------- UI events ----------------------//
void ArmModeAfterManual(TradeMode mode)
{
  g_mode = mode;
  g_last_signal_time = 0;

  SetStatus(ModeToText() + " | Manual first trade placed | SLhits=" + IntegerToString(g_sl_hits));
}

void OnChartEvent(const int id, const long &lparam, const double &dparam, const string &sparam)
{
  if(id != CHARTEVENT_OBJECT_CLICK) return;

  if(sparam == OBJ_BUY)
  {
    // First trade NOW (manual trigger), then arm BUY mode
    bool ok = PlaceFirstManualTrade(true);
    if(ok) ArmModeAfterManual(MODE_BUY_ONLY);
  }
  else if(sparam == OBJ_SELL)
  {
    bool ok = PlaceFirstManualTrade(false);
    if(ok) ArmModeAfterManual(MODE_SELL_ONLY);
  }
  else if(sparam == OBJ_OFF)
  {
    g_mode = MODE_OFF;
    SetStatus("MODE OFF (auto disabled).");
  }
}
```