```
//+------------------------------------------------------------------+
//|  Risk One-Click Panel EA (MT5)                                   |
//|  Single panel: [SELL] [Risk €] [BUY]                             |
//|  SL at previous candle wick + monetary buffer (included in Risk)  |
//+------------------------------------------------------------------+
#property strict

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

//---------------------- Inputs ----------------------//
input double InpDefaultRiskEUR   = 50.0;   // Default Risk € shown in panel
input double InpBufferEUR        = 5.0;    // Buffer in € (included within Risk €)
input bool   InpAllowMultiplePos = false; // Allow multiple positions on same symbol

//---------------------- UI names ----------------------//
string UI_PREFIX;
string OBJ_PANEL_BG, OBJ_SELL_BTN, OBJ_BUY_BTN, OBJ_RISK_EDIT,
OBJ_RISK_LBL, OBJ_STATUS_LBL;

//---------------------- 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;
}

// Estimate money-per-price-unit per 1 lot in deposit currency (EUR)
// k = (money per tick) / (tick size) => money per 1.0 price move for 1 lot
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 broker reports 0 tick_value, estimate using OrderCalcProfit for a 1-tick move
  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);
}

// Normalize volume down to step (so we don't exceed risk)
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;

  // Floor to step
  v = MathFloor(v / step) * step;

  // Clamp
  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);
}

// Normalize price to tick size (and 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;
}

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

//---------------------- Trading core ----------------------//
bool PlaceTrade(const bool isBuy)
{
   const string sym = _Symbol;
   const ENUM_TIMEFRAMES tf = (ENUM_TIMEFRAMES)_Period;

   // Permissions
   if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
   {
      SetStatus("Trading no permitido en el terminal.");
      return false;
   }
   if(!MQLInfoInteger(MQL_TRADE_ALLOWED))
   {
      SetStatus("Trading no permitido para este EA (check opciones).");
      return false;
   }

   if(!InpAllowMultiplePos && PositionSelect(sym))
   {
      SetStatus("Ya hay una posición abierta en este símbolo.");
      return false;
   }

   // Need bars
   if(Bars(sym, tf) < 3)
   {
      SetStatus("Falta histórico para calcular (necesito velas).");
      return false;
   }

   // Previous candle extremes (shift 1)
   double prevHigh = iHigh(sym, tf, 1);
   double prevLow  = iLow(sym, tf, 1);

   double bid=0, ask=0;
   if(!SymbolInfoDouble(sym, SYMBOL_BID, bid) || !SymbolInfoDouble(sym, SYMBOL_ASK,
ask))
   {
      SetStatus("No puedo leer BID/ASK.");
```

```
  return false;
}

double entry   = isBuy ? ask : bid;
double extreme = isBuy ? prevLow : prevHigh;

// Base distance from entry to candle extreme
double d0 = isBuy ? (entry - extreme) : (extreme - entry);
if(d0 <= 0)
{
  SetStatus("Distancia a mecha <= 0 (spread/rango).");
  return false;
}

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

if(RiskEUR <= BufferEUR)
{
  SetStatus("Risk € debe ser > Buffer €.");
  return false;
}

// k: € per 1.0 price move per 1 lot
double k=0.0;
if(!MoneyPerPriceUnit_1Lot(sym, k))
{
  SetStatus("No pude calcular tick value/tick size.");
  return false;
}

// Closed-form (buffer INCLUDED in total risk):
// TotalLoss = lot*k*d0 + BufferEUR
// lot = (RiskEUR - BufferEUR)/(k*d0)
double lot_raw = (RiskEUR - BufferEUR) / (k * d0);
if(lot_raw <= 0)
{
  SetStatus("Lot <= 0 (revisa Risk/vela/símbolo).");
  return false;
}

double lot = NormalizeVolumeDown(sym, lot_raw);
if(lot <= 0.0)
{
  SetStatus("Lot < mínimo. Sube Risk € o espera otra vela.");
  return false;
}

// Buffer distance in PRICE for this lot:
// db = BufferEUR / (lot*k)
double db = BufferEUR / (lot * k);
double sl = isBuy ? (extreme - db) : (extreme + db);
sl = NormalizePriceToTick(sym, sl);

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

// Verify approximate risk with OrderCalcProfit
double loss=0.0;
ENUM_ORDER_TYPE type = isBuy ? ORDER_TYPE_BUY : ORDER_TYPE_SELL;
if(!OrderCalcProfit(type, sym, lot, entry, sl, loss))
{
  SetStatus("No pude calcular riesgo final.");
  return false;
}
double risk_est = MathAbs(loss);

// If rounding pushed risk above target, reduce one step and recompute once
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 ? (extreme - db) : (extreme + db);
    sl  = NormalizePriceToTick(sym, sl);

    if(!OrderCalcProfit(type, sym, lot, entry, sl, loss))
    {
      SetStatus("No pude recalcular riesgo tras ajustar lote.");
      return false;
    }
    risk_est = MathAbs(loss);
  }
}

// Send order
trade.SetDeviationInPoints(20);

bool ok=false;
if(isBuy)
```

```
      ok = trade.Buy(lot, sym, entry, sl, 0.0, "RiskPanel BUY");
   else
      ok = trade.Sell(lot, sym, entry, sl, 0.0, "RiskPanel SELL");

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

   string side = isBuy ? "BUY" : "SELL";
   string msg = side + " OK | lot=" + DoubleToString(lot, 2)
              + " | SL=" + DoubleToString(sl, (int)SymbolInfoInteger(sym, SYMBOL_DIGITS))
              + " | Risk≈" + DoubleToString(risk_est, 2) + "€";
   SetStatus(msg);
   Print(msg);

   return true;
}

//---------------------- UI creation ----------------------//
void CreateUI()
{
   UI_PREFIX     = "RiskPanel_" + IntegerToString((int)ChartID()) + "_";
   OBJ_PANEL_BG  = UI_PREFIX + "BG";
   OBJ_SELL_BTN  = UI_PREFIX + "SellBtn";
   OBJ_BUY_BTN   = UI_PREFIX + "BuyBtn";
   OBJ_RISK_EDIT = UI_PREFIX + "RiskEdit";
   OBJ_RISK_LBL  = UI_PREFIX + "RiskLbl";
   OBJ_STATUS_LBL= UI_PREFIX + "StatusLbl";

   // Panel geometry
   int x=18, y=20;
   int panel_w=260;
   int panel_h=58;     // 1 panel, 2 lines (buttons+edit, status line)

   int top_h=28;
   int gap=6;

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

   // Background panel (single rectangle)
   ObjectCreate(0, OBJ_PANEL_BG, OBJ_RECTANGLE_LABEL, 0, 0, 0);
   ObjectSetInteger(0, OBJ_PANEL_BG, OBJPROP_CORNER, CORNER_LEFT_UPPER);
   ObjectSetInteger(0, OBJ_PANEL_BG, OBJPROP_XDISTANCE, x);
   ObjectSetInteger(0, OBJ_PANEL_BG, OBJPROP_YDISTANCE, y);
   ObjectSetInteger(0, OBJ_PANEL_BG, OBJPROP_XSIZE, panel_w);
   ObjectSetInteger(0, OBJ_PANEL_BG, OBJPROP_YSIZE, panel_h);
   ObjectSetInteger(0, OBJ_PANEL_BG, OBJPROP_BACK, false);
```

```
  ObjectSetInteger(0, OBJ_PANEL_BG, OBJPROP_COLOR, clrBlack);
  ObjectSetInteger(0, OBJ_PANEL_BG, OBJPROP_BGCOLOR, (color)0x1A1A1A); // dark
  ObjectSetInteger(0, OBJ_PANEL_BG, OBJPROP_BORDER_TYPE, BORDER_FLAT);

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

  // Risk edit (middle)
  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);

  // Risk label "€" under/over edit (small)
  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, "€");

  // BUY button (right)
  int buy_x = edit_x + edit_w + 18; // 18 includes the "€" label space
  ObjectCreate(0, OBJ_BUY_BTN, OBJ_BUTTON, 0, 0, 0);
  ObjectSetInteger(0, OBJ_BUY_BTN, OBJPROP_CORNER, CORNER_LEFT_UPPER);
  ObjectSetInteger(0, OBJ_BUY_BTN, OBJPROP_XDISTANCE, buy_x);
  ObjectSetInteger(0, OBJ_BUY_BTN, OBJPROP_YDISTANCE, row_y);
  ObjectSetInteger(0, OBJ_BUY_BTN, OBJPROP_XSIZE, btn_w);
  ObjectSetInteger(0, OBJ_BUY_BTN, OBJPROP_YSIZE, top_h);
  ObjectSetString (0, OBJ_BUY_BTN, OBJPROP_TEXT, "BUY");
  ObjectSetInteger(0, OBJ_BUY_BTN, OBJPROP_COLOR, clrWhite);
  ObjectSetInteger(0, OBJ_BUY_BTN, OBJPROP_BGCOLOR, clrDodgerBlue);

  // Status label (inside same panel, bottom line)
  ObjectCreate(0, OBJ_STATUS_LBL, OBJ_LABEL, 0, 0, 0);
  ObjectSetInteger(0, OBJ_STATUS_LBL, OBJPROP_CORNER, CORNER_LEFT_UPPER);
  ObjectSetInteger(0, OBJ_STATUS_LBL, OBJPROP_XDISTANCE, x+10);
  ObjectSetInteger(0, OBJ_STATUS_LBL, OBJPROP_YDISTANCE, y+38);
  ObjectSetInteger(0, OBJ_STATUS_LBL, OBJPROP_FONTSIZE, 9);
  ObjectSetInteger(0, OBJ_STATUS_LBL, OBJPROP_COLOR, clrGainsboro);
  ObjectSetString (0, OBJ_STATUS_LBL, OBJPROP_TEXT, "Wintrade");

  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);
  }
}

//---------------------- MT5 events ----------------------//
int OnInit()
{
  CreateUI();
  return(INIT_SUCCEEDED);
}

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

void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
  if(id == CHARTEVENT_OBJECT_CLICK)
  {
    if(sparam == OBJ_BUY_BTN)
    {
      SetStatus("Calculando BUY...");
      PlaceTrade(true);
    }
    else if(sparam == OBJ_SELL_BTN)
    {
      SetStatus("Calculando SELL...");
      PlaceTrade(false);
    }
  }
}
```