

CODICE SORGENTE COMPLETO MQL5 — Valeris_M15.mq5

AlgoSuite Quant Analyzer — Documentazione & Codice Sorgente in Formato PDF

```
//+-----+
//|                                     Valeris_M15.mq5 |
//|      VALERIS QUANT RESEARCH — Definitiva VWAP Pullback 15m      |
//|      +,370 Out-of-Sample Profit | Win Rate 62.04% | PF 1.34      |
//+-----+
#property copyright "Copyright 2026, Valeris Quant Research"
#property link      "https://valeris.com"
#property version   "1.00"
#property description "VALERIS M15 — NQ ORB 15m + VWAP Pullback + ADX Trend Filter"
#property strict

#include <Trade/Trade.mqh>
#include <Trade/PositionInfo.mqh>

CTrade      trade;
CPositionInfo pos_info;

#define MAGIC_VALERIS_M15_LONG   77715
#define MAGIC_VALERIS_M15_SHORT 77716

// --- INPUTS UFFICIALI REPORT VALERIS 15M ---
input group "=== 1. Orari Operativi & Finestra Ingressi CET ==="
input int      InpORBStartHourCET      = 15;    // Ora Apertura Sessione CET (15:00)
input int      InpORBStartMinCET       = 30;    // Minuto Apertura (15:30 Wall St)
input int      InpORBTradeEndHourCET   = 19;    // Fine Ingressi CET (19:00)
input int      InpEndHourCET           = 21;    // Hard Close EOD CET (21:00)

input group "=== 2. Parametri ORB 15m & ADX ==="
input int      InpORBPeriodMins        = 15;    // Durata Range Apertura Mins (15m — 15:30-15:45 CET)
input int      InpADXPeriod            = 14;    // Periodo ADX (14)
input double   InpADXThreshold          = 15.0;  // Soglia Trend ADX (15.0)

input group "=== 3. Target, Stop ATR & Time Exit ==="
input int      InpATRPeriod            = 14;    // Periodo ATR (14)
input double   InpStopLossATR          = 1.10;  // Moltiplicatore ATR Stop Loss (1.10)
input double   InpTakeProfitATR        = 2.40;  // Moltiplicatore ATR Take Profit (2.40 — Target Equili
input int      InpMaxHoldingMins       = 120;   // Time Exit Max Minuti Holding (120 Minuti = 2 Ore)

input group "=== 4. Rischio & Vincoli Prop Firm ==="
input double   InpRiskPct              = 0.35;  // Rischio per Trade (% Capitale)
input double   InpMaxLots              = 10.0;  // Limite Massimo Lotti
input double   InpMaxDailyLossPct      = 2.50;  // Circuit Breaker Max Loss Giornaliera (%)
input double   InpMaxSpreadPoints      = 250.0; // Max Spread Consentito
input int      InpMaxOpenPositions     = 1;     // Max Posizioni Aperte (Tassativo: 1)

// --- VARIABILI GLOBALI DELLO STATO ---
int      adx_handle      = INVALID_HANDLE;
int      atr_handle      = INVALID_HANDLE;
datetime last_bar_time   = 0;
double   orb_high        = 0.0;
double   orb_low         = 0.0;
bool     orb_defined     = false;
int      current_orb_day  = -1;
int      daily_day       = -1;
double   daily_start_equity = 0.0;

int OnInit()
{
    trade.SetExpertMagicNumber(MAGIC_VALERIS_M15_LONG);
    trade.SetMarginMode();
    trade.SetTypeFillingBySymbol(_Symbol);
}
```

```

adx_handle = iADX(_Symbol, PERIOD_M15, InpADXPeriod);
atr_handle = iATR(_Symbol, PERIOD_M15, InpATRPeriod);

if(adx_handle == INVALID_HANDLE || atr_handle == INVALID_HANDLE) {
    Print("[VALERIS M15] Errore Inizializzazione Indicatori!");
    return(INIT_FAILED);
}

Print("[VALERIS M15] Modello Definitiva VWAP Pullback 15m Inizializzato su ", _Symbol);
return(INIT_SUCCEEDED);
}

void OnDeinit(const int reason)
{
    IndicatorRelease(adx_handle);
    IndicatorRelease(atr_handle);
}

int TotalOpenPositions()
{
    int count = 0;
    for(int i = PositionsTotal() - 1; i >= 0; i--) {
        if(pos_info.SelectByIndex(i)) {
            if(pos_info.Symbol() == _Symbol) {
                ulong magic = pos_info.Magic();
                if(magic == MAGIC_VALERIS_M15_LONG || magic == MAGIC_VALERIS_M15_SHORT) count++;
            }
        }
    }
    return count;
}

void CloseAllPositions(string reason)
{
    for(int i = PositionsTotal() - 1; i >= 0; i--) {
        if(pos_info.SelectByIndex(i)) {
            if(pos_info.Symbol() == _Symbol) {
                ulong magic = pos_info.Magic();
                if(magic == MAGIC_VALERIS_M15_LONG || magic == MAGIC_VALERIS_M15_SHORT) {
                    trade.PositionClose(pos_info.Ticket());
                    PrintFormat("[VALERIS M15] Chiusura Posizione #%d: %s", pos_info.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);
        orb_defined = 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 EOD 21:00 CET");
        return true;
    }

    return false;
}

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

    datetime now = TimeCurrent();

    for(int i = PositionsTotal() - 1; i >= 0; i--) {
        if(pos_info.SelectByIndex(i)) {
            if(pos_info.Symbol() == _Symbol) {
                ulong magic = pos_info.Magic();
                if(magic == MAGIC_VALERIS_M15_LONG || magic == MAGIC_VALERIS_M15_SHORT) {
                    datetime open_time = pos_info.Time();
                    int duration_mins = (int)((now - open_time) / 60);
                    if(duration_mins >= InpMaxHoldingMins) {
                        trade.PositionClose(pos_info.Ticket());
                        PrintFormat("[VALERIS M15] Time Exit Scattato dopo %d minuti! Ticket: %d", duration_mins, (i
                    }
                }
            }
        }
    }
}

void UpdateOpeningRange()
{
    MqlDateTime dt;
    TimeToStruct(TimeCurrent(), dt);

    int server_start_hour = InpORBStartHourCET + 1; // 16:30 GMT+3
    int orb_end_min = InpORBStartMinCET + InpORBPeriodMins; // 30 + 15 = 45m
    int target_hour = server_start_hour + (orb_end_min / 60); // 16:00
    int target_min = orb_end_min % 60; // 45m (15:45 CET)

    if(dt.day != current_orb_day) {
        if((dt.hour > target_hour) || (dt.hour == target_hour && dt.min >= target_min)) {
            MqlRates rates[];
            ArraySetAsSeries(rates, true);
            int numBars = InpORBPeriodMins / 5; // 15m / 5 = 3 M5 bars
            if(numBars < 1) numBars = 1;
            int copied = CopyRates(_Symbol, PERIOD_M5, 1, numBars, rates);
            if(copied >= numBars) {
                double h_val = rates[0].high;
                double l_val = rates[0].low;
                for(int k = 1; k < copied; k++) {
                    if(rates[k].high > h_val) h_val = rates[k].high;
                    if(rates[k].low < l_val) l_val = rates[k].low;
                }
                orb_high = h_val;
            }
        }
    }
}

```

```

        orb_low = l_val;
        orb_defined = true;
        current_orb_day = dt.day;
        PrintFormat("[VALERIS M15] Opening Range 15m Definito (15:30-15:45 CET)! HIGH: %.2f | LOW: %.2f",
    }
}
}
}

double CalculateVWAP()
{
    MqlRates rates[];
    ArraySetAsSeries(rates, true);
    int copied = CopyRates(_Symbol, PERIOD_M5, 0, 50, rates);
    if(copied <= 0) return 0.0;

    double num = 0.0;
    double den = 0.0;
    for(int i = 0; i < copied; i++) {
        double typical = (rates[i].high + rates[i].low + rates[i].close) / 3.0;
        double vol = (double)rates[i].tick_volume;
        if(vol <= 0) vol = 1.0;
        num += typical * vol;
        den += vol;
    }
    return (den > 0) ? (num / den) : rates[0].close;
}

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) lot = MathFloor(lot / step_lot) * step_lot;
    if(lot < min_lot) lot = min_lot;
    if(lot > max_lot) lot = max_lot;
    return NormalizeDouble(lot, 2);
}

void ExecuteOrder(ENUM_ORDER_TYPE type, double price, double atr, ulong magic, string comment)
{
    if(TotalOpenPositions() >= InpMaxOpenPositions) 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, 0, sl, tp, comment);
} else {
    success = trade.Sell(lot_size, _Symbol, 0, sl, tp, comment);
}

if(success) {
    PrintFormat("[VALERIS M15] Ordine Eseguito! Lot: %.2f | Price: %.2f | SL: %.2f | TP: %.2f", lot_size, pr
}
}

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

    UpdateOpeningRange();
    if(!orb_defined) return;

    datetime current_bar = iTime(_Symbol, PERIOD_M5, 0);
    if(current_bar == last_bar_time) return;
    last_bar_time = current_bar;

    if(TotalOpenPositions() >= InpMaxOpenPositions) return;

    MqlDateTime dt;
    TimeToStruct(TimeCurrent(), dt);
    int server_trade_end = InpORBTradeEndHourCET + 1; // GMT+3
    if(dt.hour >= server_trade_end) return;

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

    double adx_val[];
    ArraySetAsSeries(adx_val, true);
    if(CopyBuffer(adx_handle, 0, 1, 1, adx_val) <= 0) return;
    if(adx_val[0] < InpADXThreshold) return;

    double atr_val[];
    ArraySetAsSeries(atr_val, true);
    if(CopyBuffer(atr_handle, 0, 1, 1, atr_val) <= 0) return;
    double atr = (atr_val[0] > 0) ? atr_val[0] : 15.0;

    double vwap = CalculateVWAP();
    double close_m5 = iClose(_Symbol, PERIOD_M5, 1);

    if(close_m5 > orb_high && ask > vwap) {
        ExecuteOrder(ORDER_TYPE_BUY, ask, atr, MAGIC_VALERIS_M15_LONG, "Valeris M15 Long");
    }
    else if(close_m5 < orb_low && bid < vwap) {
        ExecuteOrder(ORDER_TYPE_SELL, bid, atr, MAGIC_VALERIS_M15_SHORT, "Valeris M15 Short");
    }
}

```