MQR - Complete Manual

Practical reference for the MQR language with a focus on backtesting, events, orders, positions, management loops and export to MT5.

Responsible: Vinicius Fávero
Website: rikdomchart.com.br

Content

---

1) Overview

MQR is a DSL for trading strategies with syntax inspired by C++/MQL5. The standard flow is: write strategy, run backtest, adjust parameters and, when necessary, export to MT5.

Key Point: Even without `OnBarOpenM1()`, the engine maintains TP/SL and pending execution accuracy in M1 when there are open positions/orders in M1 OHLC mode.

2) Files and Pipeline

Extension	Purpose
`.mqr`	Strategy source code
`.exr`	Strategy compiled for execution by the engine.

RECOMMENDED PIPELINE

```
1) Write the strategy in MQR
2) Validate it in the internal backtest
3) Optimize parameters
4) Revalidate in M1 OHLC
5) Export to MT5 when it makes sense
```

3) Program and Event Structure

Event	When it fires	Common usage
`OnInit()`	Once at the beginning	SetTimeFrame, initial state, buffers
`OnBarOpenM1()`	Closing each M1	Management, schedule, trailing, cleaning
`OnBarOpen()`	First tick after TF boundary	Input signals and filters

Aliases: `OnBarOpen()` is an alias for `OnBarOpen()`, and `OnBarOpenM1()` is an alias for `OnBarOpenM1()`. `OnBar()` also works as an alias for `OnBarOpen()`. They all produce the same behavior.

MT5-compatible semantics: `OnBarOpen()` fires on **first tick of the next candle** to the boundary, like in MT5. `Close[0]` = bar forming, `Close[1]` = bar completed. For confirmed signals, use offset 1.

BASE MODEL

```
input ENUM_TIMEFRAMES inpTF = PERIOD_M5;

void OnInit() {
  SetTimeFrame(inpTF);
}

void OnBarOpenM1() {
  if (Hour() >= 17 && Minute() >= 30 && PositionsTotal() > 0) {
    CloseAll();
  }
}

void OnBarOpen() {
  if (BarIndex() < 30) return;
  if (PositionsTotal() > 0) return;

  // Close[1] = completed bar, EMA(21,1) = completed bar indicator
  double ema = EMA(21, 1);
  if (Close[1] > ema) {
    // Close[0] = current price (forming bar) for execution
    Buy(1.0, Close[0] - 80, Close[0] + 120);
  }
}
```

---

4) Types, Inputs, Enums and Operators

Base types

- double, int, bool
- ENUM_TIMEFRAMES
- custom enum

Operators

- Arithmetic: + - * / %
- Comparison: == != < > <= >=
- Logical: && || !
- Assignment: = += -= *= /= ++ --

ENUM + INPUTS

```
enum ENUM_DIRECAO {
  COMPRA = 0,
  VENDA  = 1,
  AMBOS  = 2
};

input group "Risk"
input double TP = 120.0 [60, 10, 300];
input double SL = 90.0  [30, 10, 250];

input group "Execution"
input ENUM_DIRECAO Direcao = AMBOS;
input ENUM_TIMEFRAMES TF = PERIOD_M5;
```

---

5) Mathematical Functions

The MQR language offers built-in mathematical functions that operate on `double` values. They are used for rounding, limits, powers and general calculations.

Function	Parameters	Return	Description
`Abs(x)`	x:double	double	Absolute value of x
`Max(x, y)`	x, y: double	double	Largest value between x and y
`Min(x, y)`	x, y: double	double	Smallest value between x and y
`Sqrt(x)`	x:double	double	Square root of x
`Round(x)`	x:double	double	Round to the nearest integer
`Floor(x)`	x:double	double	Round down (largest integer ≤ x)
`Ceil(x)`	x:double	double	Round up (smallest integer ≥ x)
`Pow(base, exp)`	base, exp: double	double	Power: base raised to exp

USAGE EXAMPLES

```
// Time rounding
int horaFechamento = Floor(inpHora / 100);
int minutoFechamento = inpHora % 100;

// Dynamic stop with rounded ATR
double stopPts = Round(ATR(14, 0) * 1.5);

// Range size in ticks
double rangeTicks = Round(Abs(High[0] - Low[0]) / TickSize());

// Highest high of the last 3 candles
double maxHigh = Max(High[0], Max(High[1], High[2]));

// Power for manual standard deviation calculations
double diff = Close[0] - SMA(20, 0);
double diffSq = Pow(diff, 2.0);

// Square root (e.g. for normalization)
double vol = Sqrt(Pow(High[0] - Low[0], 2.0));
```

---

6) Date and Time Functions

Functions to consult the hour, minute and day of the week of the current M1 candle. Useful for operating hours filters, closures and session control.

Function	Return	Description
`Hour()`	int	Current M1 candle time (0–23)
`Minute()`	int	Current M1 candle minute (0–59)
`DayOfWeek()`	int	Day of the week (0=Sunday, 1=Monday, ..., 6=Saturday)
`IsFirstBarOfDay()`	bool	True if it is the first candle of the day on the main timeframe
`IsLastBarOfDay()`	bool	True if it is the last candle of the day on the main timeframe
`IsFirstBarOfDayTF(tf)`	bool	Same logic, but in the `tf` timeframe
`IsLastBarOfDayTF(tf)`	bool	Same logic, but in the `tf` timeframe

STANDARD TIME FILTER

```
bool horarioPermitido() {
    int hAbertura = Floor(inpHoraAbertura / 100);
    int mAbertura = inpHoraAbertura % 100;
    int hEncerramento = Floor(inpHoraEncerramento / 100);
    int mEncerramento = inpHoraEncerramento % 100;

    bool aposInicio = (Hour() > hAbertura || (Hour() == hAbertura && Minute() >= mAbertura));
    bool antesEncerramento = (Hour() < hEncerramento || (Hour() == hEncerramento && Minute() <=
mEncerramento));

    return aposInicio && antesEncerramento;
}

void OnBarOpenM1() {
    // Everything closes at 5:30 pm
    if (PositionsTotal() > 0 && (Hour() > 17 || (Hour() == 17 && Minute() >= 30))) {
        CloseAll();
    }
}

void OnBarOpen() {
    // State reset on the first candle of the day
    if (IsFirstBarOfDay()) {
        // ... reset variables
    }
}
```

PRACTICAL EXAMPLE OF VALIDATING TIMES: START, ENTRY LIMIT AND TOTAL CLOSING

```
enum ENUM_HORA {
    H0905 = 0905,
    H1600 = 1600,
    H1630 = 1630,
    H1720 = 1720
  };

  input ENUM_HORA inpHoraInicio = H0905; // can open new orders
  input ENUM_HORA inpHoraLimite = H1630; // After that, no new orders are opened
  input ENUM_HORA inpHoraFech   = H1720; // total closure

  int HoraAtual() {
    return Hour() * 100 + Minute();
  }

  bool DentroDoHorarioOrdens() {
    int agora = HoraAtual();
    return (agora >= inpHoraInicio) && (agora < inpHoraLimite);
  }

  void FecharPorHorario() {
    if (HoraAtual() >= inpHoraFech) {
      if (PositionsTotal() > 0 || OrdersTotal() > 0) {
        CloseAll();
      }
    }
  }

  void OnBarOpenM1() {
    // Ensures closing at the exact time (M1 granularity)
    FecharPorHorario();
  }

  void OnBarOpen() {
    // Also protects in the main flow
    FecharPorHorario();

    // No new entries after the cut-off time
    if (!DentroDoHorarioOrdens()) return;

    // ... input logic
  }
```

7) Asset Properties, Arrays and Utilities

7.1) Asset properties

Current financial instrument information, useful for calculating stops at points, normalizing prices and sizing lots.

Function	Return	Description
`TickSize()`	double	Smallest price variation of the asset (ex: 0.01 for mini-index, 0.5 for mini-dollar)
`TickValue()`	double	Monetary value of 1 tick per lot (ex: R$0.20 for WINFUT)
`Digits()`	int	Number of decimal places in the price
`NormalizePrice(price)`	double	Rounds the price to the valid increment of the asset

STOP BASED ON TICKS AND NORMALIZED PRICE

```
void OnBarOpen() {
    if (PositionsTotal() > 0) return;

    double p = Close[0];
    double sl = NormalizePrice(p - 100 * TickSize());
    double tp = NormalizePrice(p + 200 * TickSize());

    Buy(1.0, sl, tp);
}
```

7.2) Arrays

Support dynamic arrays to store series and collections of values.

Function	Parameters	Description
`ArrayResize(arr, size)`	arr: array, size: int	Resizes the array to the new size
`ArraySize(arr)`	arr: array	Returns the current size of the array

EXAMPLE WITH ARRAY

```
double historico[];
int contagem = 0;

void OnBarOpen() {
    contagem++;
    ArrayResize(historico, contagem);
    historico[contagem - 1] = Close[0];

    if (ArraySize(historico) >= 10) {
        // use the last 10 values
    }
}
```

7.3) Print (debug)

Prints values to the execution log for debugging during backtest.

Function	Parameters	Description
`Print(value)`	value: double or string	Prints to the execution log. Use concatenation with + to compose messages.

DEBUG WITH PRINT

```
void OnBarOpen() {
    double rsi = RSI(14, 0);
    double atr = ATR(14, 0);
    Print("RSI=" + rsi + "ATR=" + atr + "Close=" + Close[0]);
}
```

---

8) Timeframes and Multi-Timeframe (MTF)

The main timeframe is defined at `OnInit()` with `SetTimeFrame()`. MTF readings use `*TF` functions without losing the context of the main candle.

8.1) Backtest timeframe data (main context)

The series `Open[]`, `High[]`, `Low[]`, `Close[]`, `Volume[]` and `Time[]` always reflect the main timeframe configured in the strategy. This is the context where `OnBarOpen()` makes entry decisions.

Function/Series	Description
`Open[n], High[n], Low[n], Close[n]`	OHLC of main timeframe (n=0 bar forming, n=1 last completed)
`Volume[n], Time[n]`	Main context candle volume and timestamp
`BarIndex(), BarCount()`	Current index and number of loaded bars

MAIN CONTEXT READING

```
void OnBarOpen() {
  if (BarIndex() < 30) return;

  // Close[1] = completed bar, Close[2] = previous to it
  double c1 = Close[1];
  double c2 = Close[2];
  double rangeCompletada = High[1] - Low[1];

  // Example: simple acceleration on the completed bar
  if (c1 > c2 && rangeCompletada > TickSize() * 20) {
    Print("Signal on main TF");
  }
}
```

8.2) Data from other timeframes

To query external context (H1, H4, D1, etc.), use `*TF` family: `OpenTF`, `EMATF`, `RSITF`, etc. This avoids rebuilding candles manually.

Category	Functions
OHLC/volume/time	`OpenTF`, `HighTF`, `LowTF`, `VolumeTF`, `TimeTF`
Indicators	`SMATF`, `EMATF`, `RSITF`, `ATRTF`, `MACDTF`, `MACDSignalTF`, `MACDHistTF`, `BollingerUpperTF`, `BollingerLowerTF`, `StochKTF`, `StochDTF`, `KeltnerUpperTF`, `KeltnerLowerTF`
Bar status	`BarCountTF`, `IsBarCompleteTF`, `IsNewBarTF`

INLET IN M5 WITH H1 STRUCTURAL FILTER

```
void OnBarOpen() {
  // Backtest context (M5, for example)
  double closeLocal = Close[0];

  // External context (H1)
  double openH1 = OpenTF(PERIOD_H1, 0);
  double emaH1 = EMATF(PERIOD_H1, 34, 0);

  if (openH1 > emaH1 && closeLocal > EMA(9, 0) && PositionsTotal() == 0) {
    Buy(1.0, closeLocal - 90, closeLocal + 180);
  }
}
```

8.3) How to work with indicators in the same code

A practical approach is to separate: local context for trigger and external context for direction. Common examples:

- Macro direction on H1/H4 with `EMATF`.
- Timing of entry into the main TF with `RSI` or crossing local averages.
- Volatility stop sizing with `ATR` (local) or `ATRTF` (external).

PRACTICAL COMBINATION: H1 TREND + LOCAL TIMING + ATR STOP

```
void OnBarOpen() {
  if (BarIndex() < 60 || PositionsTotal() > 0) return;

  double trend = EMATF(PERIOD_H1, 50, 0);
  double openH1 = OpenTF(PERIOD_H1, 0);
  double rsiLocal = RSI(14, 0);
  double atrLocal = ATR(14, 0);
  double p = Close[0];

  if (openH1 > trend && rsiLocal < 35) {
    Buy(1.0, p - atrLocal * 1.5, p + atrLocal * 3.0);
  }
}
```

8.4) Practical difference: local data vs external data

Quick rule: `Close[0]` is the **bar in formation** (forming bar) of the strategy timeframe — same concept as MT5. `Close[1]` is the last completed bar. `OpenTF(PERIOD_H1, 0)` returns the Open of H1, even if your strategy runs on M1/M5.

H1 TREND FILTER IN M5 STRATEGY

```
void OnBarOpen() {
  double trendH1 = EMATF(PERIOD_H1, 34, 0);
  double openH1 = OpenTF(PERIOD_H1, 0);

  if (openH1 > trendH1 && RSI(14, 0) < 35 && PositionsTotal() == 0) {
    double p = Close[0];
    Buy(1.0, p - 90, p + 180);
  }
}
```

8.5) Triple timeframe confluence (M5 + H1 + H4)

Example of local entry in the backtest timeframe with double external confirmation. This pattern reduces signals against macro direction.

TREND CONFLUENCE ON THREE HORIZONS

```
void OnBarOpen() {
  if (PositionsTotal() > 0 || BarIndex() < 80) return;

  double p = Close[0];                  // Backtest TF
  double emaM5 = EMA(21, 0);            // local indicator
  double emaH1 = EMATF(PERIOD_H1, 34, 0);
  double openH1 = OpenTF(PERIOD_H1, 0);
  double emaH4 = EMATF(PERIOD_H4, 55, 0);
  double openH4 = OpenTF(PERIOD_H4, 0);

  bool macroAlta = (openH1 > emaH1) && (openH4 > emaH4);
  bool gatilhoLocal = p > emaM5 && RSI(14, 0) < 45;

  if (macroAlta && gatilhoLocal) {
    Buy(1.0, p - 100, p + 220);
  }
}
```

8.6) Local RSI with external RSI (avoid exhaust entry)

In this pattern, the trigger happens on the main TF, but the RSI of H1 validates whether the external movement is already stretched.

LOCAL RSI + RSI H1

```
void OnBarOpen() {
  if (PositionsTotal() > 0 || BarIndex() < 50) return;

  double rsiM5 = RSI(14, 0);                    // location
  double rsiH1 = RSITF(PERIOD_H1, 14, 0);       // external
  double p = Close[0];

  // Example reading: buy only with local pullback and H1 still healthy
  if (rsiM5 < 35 && rsiH1 > 45 && rsiH1 < 70) {
    Buy(1.0, p - 90, p + 170);
  }
}
```

8.7) Synchronization by new bar from another timeframe

Use `IsNewBarTF()` to update context variables only when the external TF bar closes, avoiding recalculating everything on every local candle.

H1 CONTEXT UPDATE ONLY ON H1 CLOSURE

```
double contextoH1 = 0.0;

void OnBarOpenM1() {
  if (IsNewBarTF(PERIOD_H1)) {
    double oH1 = OpenTF(PERIOD_H1, 0);
    double eH1 = EMATF(PERIOD_H1, 34, 0);
    contextoH1 = oH1 - eH1; // positive = above average
  }
}

void OnBarOpen() {
  if (PositionsTotal() > 0) return;
  if (contextoH1 > 0 && RSI(14, 0) < 40) {
    double p = Close[0];
    Buy(1.0, p - 80, p + 160);
  }
}
```

8.8) External ATR for dynamic stop

When the main TF is very short, using ATR of H1/H4 can stabilize the stop size in intraday noise scenarios.

STOP BASED ON H1 ATR

```
input double FatorSL = 1.2;
input double FatorTP = 2.4;

void OnBarOpen() {
  if (PositionsTotal() > 0 || BarIndex() < 80) return;

  double atrH1 = ATRTF(PERIOD_H1, 14, 0);
  double p = Close[0];

  if (OpenTF(PERIOD_H1, 0) > EMATF(PERIOD_H1, 50, 0) && RSI(14, 0) < 38) {
    double sl = p - atrH1 * FatorSL;
    double tp = p + atrH1 * FatorTP;
    Buy(1.0, sl, tp);
  }
}
```

8.9) Example of combined reading of OHL in external TF

Simple example of candle amplitude on H1 used as a filter for local entry in the backtest timeframe. We only use Open, High and Low of the external TF.

H1 AMPLITUDE FILTER

```
bool CandleAmploH1() {
  double o = OpenTF(PERIOD_H1, 0);
  double h = HighTF(PERIOD_H1, 0);
  double l = LowTF(PERIOD_H1, 0);

  double range = h - l;
  if (range <= 0) return false;

  // Opening in lower half suggests buying pressure
  return (o - l) / range <= 0.4;
}

void OnBarOpen() {
  if (PositionsTotal() > 0) return;
  if (CandleAmploH1() && RSI(14, 0) < 45) {
    double p = Close[0];
    Buy(1.0, p - 85, p + 170);
  }
}
```

9) Technical Indicators

The MQR language offers built-in technical indicators with lazy precomputation and O(1) lookup per bar. They all accept an optional `offset` parameter (default 0), where 0 = forming bar, 1 = last completed bar, etc.

9.1) Basic Indicators

Function	Parameters	Return	Description
`SMA(period, offset)`	period=number of bars, offset=displacement (0=forming bar)	double	Simple Moving Average
`EMA(period, offset)`	period=number of bars, offset=displacement	double	Exponential Moving Average
`RSI(period, offset)`	period=number of bars, offset=displacement	double (0–100)	Relative Strength Index
`ATR(period, offset)`	period=number of bars, offset=displacement	double	Average True Range (volatility)

CLASSIC AVERAGE CROSSOVER

```
void OnBarOpen() {
    if (PositionsTotal() > 0 || BarIndex() < 30) return;

    // Completed bars: offset 1 and 2
    double emaRapida1 = EMA(9, 1);
    double emaRapida2 = EMA(9, 2);
    double emaLenta1  = EMA(21, 1);
    double emaLenta2  = EMA(21, 2);
    double p = Close[0]; // current price for execution

    // Upward crossover (confirmed signal)
    if (emaRapida2 < emaLenta2 && emaRapida1 >= emaLenta1) {
        double sl = p - ATR(14, 1) * 1.5;
        double tp = p + ATR(14, 1) * 3.0;
        Buy(1.0, sl, tp);
    }

    // Down crossover (confirmed signal)
    if (emaRapida2 > emaLenta2 && emaRapida1 <= emaLenta1) {
        double sl = p + ATR(14, 1) * 1.5;
        double tp = p - ATR(14, 1) * 3.0;
        Sell(1.0, sl, tp);
    }
}
```

9.2) MACD (Moving Average Convergence Divergence)

Function	Parameters	Return
`MACD(fast, slow, signal, offset)`	fast=fast EMA period, slow=slow EMA period, signal=signal period, offset=displacement (0=forming bar)	MACD Line (fast EMA − slow EMA)
`MACDSignal(fast, slow, signal, offset)`	Same parameters	Signal Line (MACD EMA)
`MACDHist(fast, slow, signal, offset)`	Same parameters	Histogram (MACD − Signal)

Notice: the `offset` parameter is optional (default 0). The three functions share the same precomputation — the first call calculates everything, the others read from the cache.

CLASSIC MACD CROSSOVER

```
void OnBarOpen() {
  if (PositionsTotal() > 0) return;

  double hist0 = MACDHist(12, 26, 9, 0);
  double hist1 = MACDHist(12, 26, 9, 1);

  double p = Close[0];
  if (hist1 < 0 && hist0 >= 0) {
    Buy(1.0, p - 100, p + 200);
  }
  if (hist1 > 0 && hist0 <= 0) {
    Sell(1.0, p + 100, p - 200);
  }
}
```

9.3) Bollinger Bands

Function	Parameters	Return
`BollingerUpper(period, mult, offset)`	period=SMA period, mult=deviation multiplier (ex: 2.0), offset=displacement	Upper band
`BollingerLower(period, mult, offset)`	Same parameters	Lower band

The middle band is simply `SMA(period, offset)`. The multiplier accepts decimal values (1.5, 2.0, 2.5 etc).

REVERSAL IN BOLLINGER BANDS

```
input int inpBBPeriod = 20;
input double inpBBMult = 2.0;

void OnBarOpen() {
  if (PositionsTotal() > 0) return;

  double upper = BollingerUpper(inpBBPeriod, inpBBMult, 0);
  double lower = BollingerLower(inpBBPeriod, inpBBMult, 0);
  double p = Close[0];

  if (p <= lower) {
    Buy(1.0, p - 80, p + 160);
  }
  if (p >= upper) {
    Sell(1.0, p + 80, p - 160);
  }
}
```

9.4) Stochastic (Stochastic Oscillator)

Function	Parameters	Return
`StochK(kPeriod, dPeriod, offset)`	kPeriod=window %K, dPeriod=smoothing %D, offset=shift	%K line (0–100)
`StochD(kPeriod, dPeriod, offset)`	Same parameters	Line %D (SMA of %K)

STOCHASTIC OVERSOLD ENTRY

```
void OnBarOpen() {
  if (PositionsTotal() > 0) return;

  double k0 = StochK(14, 3, 0);
  double d0 = StochD(14, 3, 0);
  double k1 = StochK(14, 3, 1);

  double p = Close[0];
  // Buy: %K crosses %D upwards from oversold
  if (k1 < d0 && k0 >= d0 && k0 < 30) {
    Buy(1.0, p - 90, p + 180);
  }
}
```

9.5) Keltner Channels

Function	Parameters	Return
`KeltnerUpper(emaPeriod, atrPeriod, mult, offset)`	emaPeriod=EMA of the price, atrPeriod=ATR, mult=multiplier, offset=displacement	Upper band (EMA + mult×ATR)
`KeltnerLower(emaPeriod, atrPeriod, mult, offset)`	Same parameters	Lower band (EMA − mult×ATR)

The centerline is `EMA(emaPeriod, offset)`. The multiplier accepts decimals.

BOLLINGER SQUEEZE (BOLLINGER WITHIN KELTNER)

```
void OnBarOpen() {
  double bbUpper = BollingerUpper(20, 2.0, 0);
  double bbLower = BollingerLower(20, 2.0, 0);
  double ktUpper = KeltnerUpper(20, 10, 1.5, 0);
  double ktLower = KeltnerLower(20, 10, 1.5, 0);

  // Squeeze: Bollinger bands WITHIN Keltner bands
  bool squeeze = (bbUpper < ktUpper) && (bbLower > ktLower);

  if (squeeze && PositionsTotal() == 0) {
    double hist = MACDHist(12, 26, 9, 0);
    double p = Close[0];
    if (hist > 0) Buy(1.0, p - 100, p + 200);
    if (hist < 0) Sell(1.0, p + 100, p - 200);
  }
}
```

9.6) Multi-Timeframe Variants of Indicators

All indicators have MTF variants with the suffix `TF`, which receive the timeframe as the first parameter:

Local function	MTF equivalent
`SMA(21,0)`	`SMATF(PERIOD_H1,21,0)`
`EMA(21,0)`	`EMATF(PERIOD_H1,21,0)`
`RSI(14,0)`	`RSITF(PERIOD_H1,14,0)`
`ATR(14,0)`	`ATRTF(PERIOD_H1,14,0)`
`MACD(12,26,9,0)`	`MACDTF(PERIOD_H1,12,26,9,0)`
`MACDSignal(12,26,9,0)`	`MACDSignalTF(PERIOD_H1,12,26,9,0)`
`MACDHist(12,26,9,0)`	`MACDHistTF(PERIOD_H1,12,26,9,0)`
`BollingerUpper(20,2.0,0)`	`BollingerUpperTF(PERIOD_H1,20,2.0,0)`
`BollingerLower(20,2.0,0)`	`BollingerLowerTF(PERIOD_H1,20,2.0,0)`
`StochK(14,3,0)`	`StochKTF(PERIOD_H1,14,3,0)`
`StochD(14,3,0)`	`StochDTF(PERIOD_H1,14,3,0)`
`KeltnerUpper(20,10,1.5,0)`	`KeltnerUpperTF(PERIOD_H1,20,10,1.5,0)`
`KeltnerLower(20,10,1.5,0)`	`KeltnerLowerTF(PERIOD_H1,20,10,1.5,0)`

---

10) Trade: Orders, Positions and Tickets

Group	Main functions
Opening	`Buy`, `Sell`, `BuyLimit`, `SellLimit`, `BuyStop`, `SellStop`
Management	`PositionModify`, `OrderModify`, `ClosePosition`, `CloseAll`, `OrderDelete`
Position query	`PositionsTotal`, `PositionTicket`, `PositionEntry`, `PositionSL`, `PositionTP`, `PositionLots`, `PositionProfit`, `PositionType`
Order query	`OrdersTotal`, `OrderTicket`, `OrderPrice`, `OrderSL`, `OrderTP`, `OrderLots`, `OrderType`

---

11) Loops and Position/Order Metrics (advanced)

This section covers exactly the loop patterns to cycle through open orders/positions and extract useful management metrics.

11.1) Position loop with total PnL and exposure

SUMMATION PER LOOP

```
void OnBarOpenM1() {
  double pnlTotal = 0.0;
  double lotsTotal = 0.0;

  for (int i = 0; i < PositionsTotal(); i++) {
    double tk = PositionTicket(i);
    pnlTotal += PositionProfit(tk);
    lotsTotal += PositionLots(tk);
  }

  Print("PnL=" + pnlTotal + "| Lots=" + lotsTotal);
}
```

11.2) Weighted average entry price (all positions)

AVERAGE PRICE PER LOT

```
double PrecoMedioPosicoes() {
  double somaPxLots = 0.0;
  double somaLots = 0.0;

  for (int i = 0; i < PositionsTotal(); i++) {
    double tk = PositionTicket(i);
    double lots = PositionLots(tk);
    double px = PositionEntry(tk);

    somaPxLots += px * lots;
    somaLots += lots;
  }

  if (somaLots <= 0) return 0.0;
  return somaPxLots / somaLots;
}

void OnBarOpenM1() {
  if (PositionsTotal() > 0) {
    double pm = PrecoMedioPosicoes();
    Print("Average price of wallets=" + pm);
  }
}
```

11.3) Average price separated by direction (buy/sell)

AVERAGES PER SIDE

```
void MediasPorLado(double &pmBuy, double &pmSell, double &lotsBuy, double &lotsSell) {
  double somaBuy = 0.0;
  double somaSell = 0.0;
  lotsBuy = 0.0;
  lotsSell = 0.0;

  for (int i = 0; i < PositionsTotal(); i++) {
    double tk = PositionTicket(i);
    double tp = PositionType(tk);   // 0 buys, 1 sells
    double lt = PositionLots(tk);
    double pe = PositionEntry(tk);

    if (tp == 0) {
      somaBuy += pe * lt;
      lotsBuy += lt;
    } else {
      somaSell += pe * lt;
      lotsSell += lt;
    }
  }

  pmBuy = (lotsBuy > 0) ? (somaBuy / lotsBuy) : 0.0;
  pmSell = (lotsSell > 0) ? (somaSell / lotsSell) : 0.0;
}
```

11.4) Batch break-even (SL adjustment per ticket)

LOOP WITH POSITIONMODIFY

```
void OnBarOpenM1() {
  for (int i = 0; i < PositionsTotal(); i++) {
    double tk = PositionTicket(i);
    double lucro = PositionProfit(tk);

    if (lucro >= 120.0) {
      double entrada = PositionEntry(tk);
      double tp = PositionTP(tk);
      PositionModify(tk, entrada, tp); // moves SL to break-even
    }
  }
}
```

11.5) Pending order loop and average activation price

TRIGGER AVERAGE PER PENDING

```
void OnBarOpenM1() {
  double somaPx = 0.0;
  double somaLots = 0.0;

  for (int i = 0; i < OrdersTotal(); i++) {
    double tk = OrderTicket(i);
    double px = OrderPrice(tk);
    double lt = OrderLots(tk);

    somaPx += px * lt;
    somaLots += lt;
  }

  if (somaLots > 0) {
    double precoMedioPendentes = somaPx / somaLots;
    Print("Average price pending=" + precoMedioPendentes);
  }
}
```

11.6) Cancel pending issues away from the current price

ORDER BOOK HYGIENE

```
input double DistMaxTicks = 120.0;

void OnBarOpenM1() {
  double p = Close[0];
  double lim = DistMaxTicks * TickSize();

  for (int i = OrdersTotal() - 1; i >= 0; i--) {
    double tk = OrderTicket(i);
    double op = OrderPrice(tk);

    if (Abs(op - p) > lim) {
      OrderDelete(tk);
    }
  }
}
```

11.7) Aggregate closing by portfolio target

CLOSEALL BY CONSOLIDATED PNL

```
input double AlvoCarteira = 500.0;
input double StopCarteira = -300.0;

void OnBarOpenM1() {
  double pnl = 0.0;

  for (int i = 0; i < PositionsTotal(); i++) {
    double tk = PositionTicket(i);
    pnl += PositionProfit(tk);
  }

  if (pnl >= AlvoCarteira || pnl <= StopCarteira) {
    CloseAll();
  }
}
```

Important Note: When removing items in an order/position loop, prefer to iterate backwards (`for i = total-1; i >= 0; i--`) to avoid index inconsistency after deletions.

---

12) Backtest, Spread and SL-first Criterion

M1 OHLC

- Granularity per M1 candle
- More faithful for validation
- TP/SL and pending valued continuously at M1 when there are positions/orders

Input timeframe

- Faster for optimization
- Less intrabar detail
- Use for screening; final validate in M1 OHLC

Spread simulation: in the backtest, it is possible to configure spread in points. This cost affects inputs/outputs according to the simulated Bid/Ask logic.

SL-first criterion: When SL and TP could be played on the same bar, the engine assumes SL first (conservative stance).

13) MQR x MT5 (what the converter solves)

Point	MQR	MT5	Status
Bar indexing	`Close[0]` = forming bar	`iClose(0)` = forming bar	Identical (no adjustment)
Division	Double semantics	`int/int` can truncate	Converter adjusts
Modulo	Double compatible	`%` is integer-only	Converter uses `fmod`
Events	OnBarOpenM1 + OnBarOpen	OnTick	Converter mounts equivalent flow
Execution price	Engine simulation	Real Bid/Ask from tester	Inherent difference

14) Robustness Checklist

- Separate input (`OnBarOpen`) from management (`OnBarOpenM1`)
- Control risk per ticket and portfolio (aggregated PnL)
- Measure average price per lot and total exposure
- Clear pending old/away from current price
- Optimize in fast mode and validate in M1 OHLC
- Compare MQR x MT5 results considering inherent execution differences

15) Appendix: complete ENUM_TIMEFRAMES

Constant	Minutes
PERIOD_CURRENT	0
PERIOD_M1	1
PERIOD_M2	2
PERIOD_M3	3
PERIOD_M4	4
PERIOD_M5	5
PERIOD_M6	6
PERIOD_M10	10
PERIOD_M12	12
PERIOD_M15	15
PERIOD_M20	20
PERIOD_M30	30
PERIOD_H1	60
PERIOD_H2	120
PERIOD_H3	180
PERIOD_H4	240
PERIOD_H6	360
PERIOD_H8	480
PERIOD_H12	720
PERIOD_D1	1440
PERIOD_W1	10080
PERIOD_MN1	43200

16) Appendix: Position and Order Type Constants

POSITION_TYPE_*

Constant	Value	Description
POSITION_TYPE_BUY	0	Buying position
POSITION_TYPE_SELL	1	Sales position

ORDER_TYPE_*

Constant	Value	Description
ORDER_TYPE_BUY	0	Market purchase order
ORDER_TYPE_SELL	1	Market sales order
ORDER_TYPE_BUY_LIMIT	2	Buy Limit Order
ORDER_TYPE_SELL_LIMIT	3	Sell Limit Order
ORDER_TYPE_BUY_STOP	4	Buy Stop Order
ORDER_TYPE_SELL_STOP	5	Sell Stop Order

Tip: Use these constants when comparing the return of `PositionType()` and `OrderType()` instead of fixed numbers.