QuantMonitor System Documentation

Detailed explanation of the three EAs, the database architecture, the connection between backtest and live trading, and the use of the database for real-time analytics.

Important note. In this document, the “three EAs” are understood as the following set:

- QuantMonitor MT5 Monitoring.mq5
- QuantMonitor Portfolio Overlay.mq5
- QuantMonitor Multi Market Regime Dashboard v124.mq5

QM Equity Panel.mq5 is not treated here as a separate EA, but as a supporting indicator used by the first EA to draw the equity curve in a separate window.

1. Overall purpose of the system

The entire architecture is designed so that the trader always knows four things.

First, what the strategy did historically in backtest.
Second, what the strategy is doing now in live trading.
Third, how a continuous backtest-plus-live curve looks without manual rewriting.
Fourth, which market regime is currently active and whether that regime is favorable for the given strategy.

This is fully consistent with the logic of QuantMonitor Premium Playbook. The book argues that the trader should not merely collect ideas, but should assemble research, implementation, monitoring, validation, and regime awareness into a single operating framework. In that framework, trading is not one isolated skill, but a stack of connected layers in which edge, implementation, risk, validation, and monitoring must support one another.

The whole system is therefore best understood as a professional supervision framework for algorithmic trading systems rather than as a collection of visual panels.

2. System architecture in one sentence

QuantMonitor MT5 Monitoring creates and continuously refreshes the database of one strategy.
QuantMonitor Portfolio Overlay takes the resulting equity series from several such databases and assembles a portfolio view.
QuantMonitor Multi Market Regime Dashboard adds market context, that is, an answer to the question of which market regime is active now and how close that regime is to the ideal conditions for each specific strategy.

3. EA No. 1: QuantMonitor MT5 Monitoring.mq5

This is the main collection and synchronization EA. It is the data heart of the whole system.

What it does exactly

After startup, it creates the identity of the current instance from the triple symbol plus strategy comment plus comment-matching mode. From that identity, it builds a hash, and from that hash it derives the SQLite database name and the equity CSV name.

It then performs four main steps:
1. it loads the backtest from a Strategy Tester HTML report,
2. it loads live trades from MT5 history,
3. it merges backtest and live into one continuous layer,
4. it builds daily equity, statistics, and chart output from the merged data.

Main input variables

InpStrategyComment
The strategy comment. This is the key identifier of live trades. Live history is searched according to this comment.

InpBacktestFile
The file name of the HTML report exported from Strategy Tester.

InpTimerSeconds
The period of automatic refresh. The EA updates itself repeatedly.

InpCommentMatchMode
0 equals exact.
1 equals contains.

The contains mode is important when a broker or copier modifies part of the comment but you still need reasonable matching.

How data flows through the EA

First, the system builds the instance ID.
This is created from the text symbol | comment | match mode.
A simple hash is then applied, and from this the following names are derived:
QM_<instance>.sqlite
EQ_<instance>.csv

This means that every strategy on every symbol has its own separate data branch.

How backtest import works

BacktestImporter.mqh reads the complete HTML report and searches mainly for:
Symbol,
CustomComment,
Initial Deposit,
and the Deals section.

From the trade section, it reads individual rows and converts them into the internal QMDeal structure.

A very important detail is that the tester HTML report normally does not contain the same position_id values as live MT5 history. For that reason, the importer creates synthetic position IDs. It does so by tracking the state of open long and short positions separately and, on entry and exit, assigning its own synthetic_position_id values. This later makes it possible to segment backtest deals into individual trades in a way similar to live data.

Without this layer, it would not be possible to build correct entry-exit pairing, correct trade segmentation, correct profit factor and win rate, and correct daily equity.

How live trade collection works

HistoryCollector.mqh uses HistorySelect(0, TimeCurrent()) and scans historical deals.

The matching logic has two layers.

The first layer is direct comment matching.
If the comment matches the strategy comment, the position_id of that deal is stored as relevant.

The second layer is position-link matching.
Once a position_id has been recognized as relevant, all deals with the same position_id are taken, even if some of them no longer carry the exact same comment.

This is a very practical real-world solution. In live trading, it is common that the entry still has the original comment but later exit, partial close, or service deals are not textually identical. If the system relied only on direct comment text, part of the live history would break apart. The position_id connection preserves the whole trade.

How backtest and live are merged

This is the most important part of the architecture.

MergeEngine.mqh takes the arrays backtest_deals and live_deals. If live data do not exist, the system uses pure backtest. If live data do exist, it finds earliest_live_time, that is, the time of the earliest live deal. It then builds merged data as follows:
all backtest deals older than the first live deal are kept,
and from that point onward only live deals are used.

In other words, the live layer does not blindly attach new live data to the end. From the moment live starts, live replaces the historical approximation with reality.

This is exactly the correct operational behavior.
Backtest serves as the historical approximation.
Live becomes the authority from the moment it begins.

As a result, a continuous merged history is created, which is usable for equity, trade statistics, trade segmentation, portfolio overlay, and analysis of performance development over time.

How daily equity is built

QmBuildDailyEquity processes merged_deals day by day. For each deal, it takes profit plus swap plus commission, aggregates these values by day, and builds cumulative equity. The starting point is the initial_balance obtained from the backtest report.

This produces the daily_equity table, which later becomes highly valuable because basic analytics no longer need the full trade reconstruction layer.

How statistics are built

StatsEngine.mqh first builds trade segments from deals. This means it combines entries and exits into true trades according to position_id. It can also work with partial closes, because it allocates the cost component proportionally by volume.

Only after that does it calculate:
number of trades,
number of wins,
gross profit,
gross loss,
profit factor,
win rate.

The panel compares:
Backtest,
1Y,
6M,
3M.

This is a very strong design, because the trader immediately sees whether live or recent performance still behaves in line with the historical logic.

Which tables exist in the database

meta
Key-value metadata of the instance.

backtest_files
Evidence of whether the backtest HTML was found.

backtest_deals
Imported deals from Strategy Tester.

live_deals
Deals loaded from MT5 history.

merged_deals
The resulting combined backtest-plus-live layer.

daily_equity
Daily PnL and cumulative equity.

Which metadata are stored

Typical examples include:
instance_id,
symbol,
strategy_comment,
backtest_file,
comment_match_mode,
backtest_count,
live_count,
merged_count,
daily_equity_count,
initial_balance,
rebuilt_at,
last_sync_time,
last_sync_matched_count.

This is very important for auditability. If anything later looks inconsistent, it is possible to track what was synchronized and when.

Where files are stored

AppConfig.mqh defines:
DB_FOLDER_NAME = QuantMonitor
DB_FOLDER_PATH = QuantMonitor\

In practice, this means that the system uses the QuantMonitor folder inside the terminal's file area.

SQLite database of one instance:
QuantMonitor\QM_<instance>.sqlite

Equity CSV:
QuantMonitor\EQ_<instance>.csv

The equity CSV is especially important because it is written through FILE_COMMON, which allows it to be shared with the supporting indicator QM Equity Panel.

What this EA contributes

This EA does not merely display something in the chart. It creates a unified data model of one strategy. In other words, one strategy becomes a continuously updated analytical unit usable not only visually, but also at the database level.

4. EA No. 2: QuantMonitor Portfolio Overlay.mq5

The second EA is not a data collector. It is a portfolio visualization layer built on top of already existing databases.

What it does exactly

Its input is a list of strategies in text form. It accepts formats such as:
COMMENT
or
SYMBOL:COMMENT
or
SYMBOL|COMMENT

For example:
XAUUSD:XU_30_913034165_S_HH_CF_SQX, BRENT:Brent hh1, .US30Cash:New Strategy, EURJPY:EJ_15_201514104_S_SM_CF_SQ4

The EA parses this list into individual specifications. Then, for each specification, it searches the QuantMonitor folder for the most appropriate database.

How it finds the correct database

Portfolio Overlay does not work in a rigid “open exactly this file” style. Instead, it scans all SQLite files in the QuantMonitor folder and reads metadata from the meta table, especially:
symbol,
strategy_comment,
rebuilt_at,
and, where available, last_sync_time or created_at.

It then selects the database that:
matches the required symbol,
matches the required strategy comment,
and is the most recent.

This is operationally very good, because the user does not need to maintain a manual map of file names. The EA finds the correct database by itself.

Which data layers it uses

First, it tries to read the daily_equity table directly.
If that table is empty or incomplete, it falls back to rebuilding equity from merged_deals.

This makes Portfolio Overlay robust. It is not dependent on a single data representation.

How it draws the portfolio

Each strategy is assigned a color.
On the chart timeline, it draws a polyline from the equity points.
It can also create point tooltips so that the user can hover over a date and see equity and daily PnL.

At the same time, it builds a portfolio total. It does this by accumulating the daily PnL of all series into a common portfolio_daily array and then constructing cumulative portfolio equity from it.

Visually, the trader sees:
separate strategy equity curves,
the aggregated portfolio curve,
and a legend with realized trade counts and database availability.

Important input variables

InpStrategyComments
A comma-separated list of strategies.

InpCommentMatchMode
The same exact or contains logic.

InpDaysBack
How many days of history should be displayed.

InpTimerSeconds
How often the overlay should refresh.

InpHidePriceChart
If true, the candle chart becomes visually hidden and only the overlay remains.

InpLineWidth
The thickness of the lines.

InpShowPointTooltips
Whether point tooltips should be displayed.

Why this EA matters

It is a fast portfolio supervision layer. The trader immediately sees:
which strategy is driving the portfolio,
which strategy is dead or flat,
whether the portfolio is made of truly different curves,
and whether diversification is real or only imagined.

This fits very well with the part of the book dealing with professional portfolio construction. The book stresses that a portfolio should not rely on one market logic alone, but on multiple edge families, multiple failure modes, and multiple opportunity regimes. It also warns that complexity should grow more slowly than understanding. That is exactly why Portfolio Overlay makes sense only after every individual strategy already has its own clean database and monitoring layer.

5. EA No. 3: QuantMonitor Multi Market Regime Dashboard v124.mq5

The third EA is not about trade history. It is about the current market and whether today’s market corresponds to the ideal conditions of each strategy.

From the perspective of trading philosophy, this is the most important layer of understanding.

What it does exactly

For each defined strategy, it takes:
the symbol,
the strategy comment,
the strategy profile from CSV,
and current market data for that symbol.

It then calculates a market snapshot and compares it with the ideal profile of the strategy.

The result is a score from 0 to 100 and a state label:
FAVORABLE,
NEUTRAL,
UNFAVORABLE.

This means that the trader no longer watches only the result of the strategy, but also the context in which the strategy is operating now.

Which inputs it uses

InpStrategyComments
A list of strategies in SYMBOL:COMMENT format.

InpProfileCsv
The path to the CSV file containing ideal profiles.

InpRegimeTimeframe
The timeframe on which the regime is calculated, H1 in your case.

InpTimerSeconds
Panel refresh period.

InpLookbackBars
The number of bars used for regime calculation and especially for percentile evaluation of volatility.

There are also a number of layout-related variables such as:
InpPanelX,
InpPanelY,
InpMaxColumns,
InpShowLegend,
InpShowNumericRows,
InpBlackoutCanvas,
InpRowHeight,
and similar controls.

How the market regime is calculated

MarketRegimeEngine_v121.mqh calculates the following values:
close,
EMA50,
EMA200,
ATR14,
ATR50,
ATR percent,
ADX14,
RSI14,
ER24,
EMA50 slope normalized by ATR,
and MA gap in ATR units.

From that, it derives the following state variables:

Direction
UP, DOWN, SIDE

Structure
TREND or CHOP

Volatility
LOW, MID, HIGH

Phase
COMPRESSION, NORMAL, EXPANSION

Momentum
BULL, NEUTRAL, BEAR

Trend strength
WEAK, MODERATE, STRONG

Exact logic

Direction
UP when EMA50 is above EMA200 and slope normalized is at least 0.12 and MA gap is at least 0.50.
DOWN in the opposite case.
Otherwise SIDE.

Structure
TREND if ADX is at least 25 and ER24 is at least 0.30.
Otherwise CHOP.

Volatility
Based on ATR-percent percentiles.
Up to the 33rd percentile is LOW, above the 67th percentile is HIGH, and between them is MID.

Phase
ATR14 divided by ATR50.
Below 0.90 equals COMPRESSION.
Above 1.10 equals EXPANSION.
Otherwise NORMAL.

Momentum
RSI at least 60 equals BULL.
RSI at most 40 equals BEAR.
Otherwise NEUTRAL.

Trend strength
ADX below 18 equals WEAK.
ADX below 30 equals MODERATE.
Otherwise STRONG.

How the strategy profile CSV works

The CSV does not contain only one sentence such as “this strategy likes trend.” It contains two layers of ideal conditions.

The first layer is categorical:
best_dir_state,

best_structure,
best_vol_state,
best_phase,
best_momentum,
best_trend_strength.

The second layer is numerical:
for ADX, ATR percent, ER24, RSI14, slope, and MA gap, the file stores q25, median, and q75.

This is a very strong design.
The categories define which general type of market the strategy prefers.
The numerical quartile bands define the value ranges within which the strategy historically performed best.

How the match score is calculated

The score is divided into two halves:
60 points for categorical matches,
40 points for numerical values that lie inside the ideal interquartile band.

Each of the six categorical dimensions has a weight of 10 points.
Each of the six numerical variables has a weight of 40 divided by 6.

The final labels are:
70 and above equals FAVORABLE,
40 to 69.99 equals NEUTRAL,
below 40 equals UNFAVORABLE.

That is the correct design because the system is not a binary on or off switch. It is a measure of how close the current market is to the ideal market for the given EA.

Why this EA matters

This is the layer that answers the question:
Is the strategy currently operating in its home environment, or outside it?

This fits the book exactly. The book argues that the trader should know the regime he is in, and that regime awareness should be operational rather than mystical. It also emphasizes live system-health monitoring and the ability to distinguish a normal losing period from a loss that points to structural change.

6. How the database connects backtest and live exactly

This is the key point of the whole platform.

Backtest is imported from the HTML report as backtest_deals.
Live is collected from MT5 history as live_deals.
The merge layer creates merged_deals in the following way:
history before the first live deal remains backtest,
from the first live deal onward, live becomes the authority.

This means that the database does not store backtest and live as unrelated layers, but as one connected timeline of a single strategy.

The practical meaning is enormous:
if live does not yet exist, the strategy still has full backtest history;
when live begins, reality takes over from a defined point;
equity becomes continuous;

statistics for one year, six months, and three months begin to reflect real operation;
Portfolio Overlay can combine an older backtest foundation with newer live reality;
and the regime dashboard can be interpreted against actual strategy performance.

The database is therefore not merely an archive. It is a live interface between research and execution.

7. How the database can be used for real-time analytics

This is where the real potential of the system appears.

The simplest use is visual.
The monitoring EA shows trades, equity, and core statistics.
Portfolio Overlay shows the combined equity of multiple strategies.
The regime dashboard shows whether the current market is favorable or unfavorable.

But the database allows much more.

It can support:
comparison of backtest profit factor versus live profit factor;
comparison of backtest win rate versus the last year, half-year, and last three months;
tracking the deviation between expected and realized behavior;
tracking changes in performance after volatility changes;
tracking whether a strategy fails in a specific type of market;
tracking whether the portfolio is overloaded with one regime type.

In practice, this means that the database can become a live analytical layer for capital allocation. It can answer not only “how much did the strategy make,” but also “in which market did it make money,” “in which market does it lose,” and “is the current market similar to its ideal state.”

This is very close to what the book describes as the move from fragments to a framework: the trader should understand not only the setup, but the full operating environment of edge, validation, implementation, risk, and monitoring.

8. Why the system was built this way according to the book’s logic

According to the book, a professional trader must do several things at once:
have a reason for the edge,
translate the idea into rules,
validate it adversarially,
know the market regime,
monitor system health,
understand the difference between research, trading, and review,
and be able to explain why a strategy is making or losing money.

This architecture does exactly that.

The Monitoring EA is responsible for the transition from research to executable and monitored implementation.
Portfolio Overlay is responsible for the portfolio layer and the supervision of several edge families.
The Regime Dashboard is responsible for operational regime reading.

In other words, the system exists so that the trader is not a blind operator of EAs, but the manager of his own edge.

The logic of the book is explicit on this point: the trader should know what he is trading, when he is trading it, and why he is trading it. He should translate ideas into rules early, know the regime he is in, monitor system health, and be able to explain why a strategy made or lost money.

9. Explanation of the most important variables and files

AppConfig.mqh
The basic project configuration. It defines the application name and the QuantMonitor data folder.

Models.mqh
The core data structures:
QMDeal,
QMDailyEquityPoint,
QMSyncStats.

Db.mqh
SQLite work:
opening the database,
creating the schema,
insert and replace into tables,
metadata handling.

HistoryCollector.mqh
Loads live history from MT5 and matches it by symbol, comment, and position_id.

BacktestImporter.mqh
Parses the HTML tester report and converts it into internal deals.

MergeEngine.mqh
Combines backtest and live into one time layer.

StatsEngine.mqh
Builds trade segments and calculates PF, WR, and period-based statistics.

ChartRenderer.mqh
Draws trades and statistics into the chart.

QM Equity Panel.mq5
An indicator for a separate window. It reads the equity CSV from common files and draws the equity curve.

QuantMonitor MT5 Monitoring.mq5
The main synchronization and database EA.

QuantMonitor Portfolio Overlay.mq5
Portfolio visualization across multiple databases.

MarketRegimeEngine_v121.mqh
The calculation of market regimes and loading of ideal strategy profiles from CSV.

QuantMonitor Multi Market Regime Dashboard v124.mq5
The multi-strategy regime panel over one chart.

QM_MarketRegimeProfiles.csv
The database of ideal market profiles for the strategies.

10. Practical file paths

From the user's perspective, it is best to keep the following structure:

MQL5\Experts\
for the EA files.

MQL5\Include\QuantMonitor\
for the .mqh modules.

MQL5\Files\QuantMonitor\
for local CSV and local read files.

Common\Files\QuantMonitor\
for equity CSV used by QM Equity Panel and for shared files.

QuantMonitor\QM_<instance>.sqlite
the database of one specific strategy.

QuantMonitor\EQ_<instance>.csv
the equity CSV of one specific strategy.

QuantMonitor\QM_MarketRegimeProfiles.csv
the CSV file with ideal regime profiles.

11. How to use the system correctly in practice

The recommended workflow is this.

First, export the backtest HTML.

Then run QuantMonitor MT5 Monitoring for the specific strategy.

Check that it found the backtest, loaded live, and created the database.

Then run Portfolio Overlay, which takes multiple equity series from the databases.

Finally, run Market Regime Dashboard, which tells you how favorable the current market is for the defined strategies.

The correct interpretation is then not only "the strategy made money today" or "the strategy lost money today," but rather:
the strategy is doing well because it is in a favorable regime;
the strategy is doing poorly, but the regime is historically bad for it, so the result is normal;
the strategy is doing poorly and the market score is low, so this is not necessarily edge failure but a mismatch with the environment;
the strategy is doing poorly even in a regime that was historically favorable, and that is a signal for deeper review.

This is exactly the type of understanding the book treats as professional: not only the result, but also the process, the regime, the validity, and the explanation of the result.

12. Brief conclusion

Taken together, the three EAs form a highly coherent supervisory system.

QuantMonitor MT5 Monitoring creates the data truth of one strategy.
Portfolio Overlay creates the portfolio truth of several strategies.
Market Regime Dashboard creates the contextual truth of the market regime in which those strategies are operating.

By connecting backtest and live data inside one database, the system creates a living analytical layer that can be used not only for visualization, but also for ongoing evaluation of robustness, system health, and suitability of the current market.

That is exactly the professional way of working promoted by the book: the trader should understand edge, implementation, regime, validation, and the reasons for profit and loss, rather than mechanically running an EA.