How the ay-MarketProfileDWM_V1p40_ENB.mq4 Indicator works

This MQL4 indicator is designed to display a Market Profile on the chart. It can show profiles based on Time Price Opportunity (TPO) or actual tick volume, for various periods like daily, weekly, monthly, or intraday (H4, H1, M30, M15, M5).

Here's a breakdown of how it works:

1. Overall Purpose

The indicator visualizes where price has spent the most time (TPO) or where the most volume has been traded over a specified period. This helps traders identify significant price levels, such as the Point of Control (POC) and Value Area (VA).

2. Input Parameters (`extern` variables)

The user can customize the indicator through several input parameters, including:

- `LookBack`**: Number of past profile periods to display.**
- `UseVolumeProfile`**: If true, calculates profile based on tick volume; otherwise, uses TPO.**
- `ProfileTimeframe`**: The period each profile represents (e.g., "D" for Daily, "W" for Weekly, "H1" for 1-Hour).**
- `DayStartHour`**: For daily profiles, the hour when a new day's profile begins.**
- `VATPOPercent`**: The percentage of TPO/Volume used to calculate the Value Area (typically 70%).**
- `TickSize`**: The price increment for building the profile's price levels.**
- **Display Options: Various booleans to show/hide the price histogram, Value Area, VAH/VAL lines, POC lines, and open/close arrows.**
- **Colors & Design: Parameters to customize colors of different profile elements and the histogram appearance.**
- **Data Timeframes: Inputs to specify the lower timeframe data (e.g., M1, M5, M15) to use for constructing higher timeframe profiles (e.g.,** `DailyProfileDataTf`**,** `WeeklyProfileDataTf`**).**

3. Core MQL4 Functions

- `init()` **(Initialization Function):**

    - Called once when the indicator is first loaded or when its input parameters are changed.
    - Sets up global variables based on the input parameters.
    - Determines the `giProfileTf` (the MQL4 constant for the chosen `ProfileTimeframe`) and `giDataTf` (the MQL4 constant for the data resolution to build the profile, e.g., using M15 data for an H1 profile).
    - Calculates `fpoint` and `fdigits` to handle different broker price precisions (4/5 digit, 2/3 digit).
    - Calculates `gdOneTick` (the actual price value of one `TickSize` step) and `giStep` (how many `TickSize` units are grouped for histogram drawing).
    - Sets a unique prefix (`gsPref`) for all chart objects created by this instance of the indicator to avoid conflicts.
- `deinit()` (Deinitialization Function):
    - Called once when the indicator is removed from the chart or the chart is closed.
    - Calls `delObjs()` to remove all chart objects created by this indicator instance, cleaning up the chart.
- `start()` (Main Calculation and Drawing Function):
    - This is the heart of the indicator, called on every new tick by default.
    - Compatibility Check (`isOK()`): First, it checks if the current chart's timeframe is suitable for building the selected `ProfileTimeframe`. If not, it exits.
    - LookBack Adjustment: Ensures `LookBack` doesn't request more profiles than available bar data.
    - New Profile Bar Detection (`newBarProfileTf()`): Checks if a new bar has started for the chosen `ProfileTimeframe` (e.g., a new day for D1 profile, a new hour for H1 profile).
        1. If a new profile bar starts, it calls `delObjs()` to clear all previously drawn profiles and sets `endbar_proftf` to `LookBack-1`. This means it will redraw all `LookBack` number of profiles.
        2. If not a new profile bar, `endbar_proftf` is 0, meaning only the current, developing profile will be updated.
    - Main Profile Loop: It then loops from `endbar_proftf` down to 0 (current profile). For each `ibar_proftf` (profile index):
        1. `ArrayResize(aprice_step, 0)`: Clears the main data array for the current profile calculation.

2. `getStartAndEndBar()`: Determines the start and end bar indices on the `giDataTf` (data collection timeframe) that correspond to the `ibar_proftf` period. For example, for a daily profile, it finds the first and last M15 bars of that day.
3. `getHHLL()`: Finds the highest high and lowest low within that determined bar range for the profile.
4. `getPriceTPO()`: This is the core calculation engine:
    - It populates the `aprice_step` array with discrete price levels from the profile's low to high, stepped by `gdOneTick`.
    - It then iterates through each bar within the profile's data range (`startbar_datatf` to `endbar_datatf`).
    - For each of these bars, it checks which price levels in `aprice_step` were touched by the bar's high-low range.
    - If a price level is touched:
        - Its TPO count (`aprice_step[level][TPOIDX]`) is incremented.
        - If `UseVolumeProfile` is true, its volume (`aprice_step[level][VOLIDX]`) is incremented by a share of the bar's volume.
    - During this process, it keeps track of the total TPOs, total volume, the price level with the highest TPO (TPO POC), and the price level with the highest volume (Volume POC). Tie-breaking for POC is done by choosing the level closer to the profile's midpoint price.
5. `drawPriceHistoAndPOCLines()`: Draws the horizontal histogram bars (either TPO or volume based on `UseVolumeProfile`) and the Point of Control (POC) line. It also identifies and can color "Virgin POCs" (POCs from past profiles that haven't been touched by subsequent price action).
6. `getValueArea()`: Calculates the Value Area. It starts from the POC and expands outwards, level by level, accumulating TPOs/volume until `VATPOPercent` (e.g., 70%) of the total TPO/volume for that profile is included. The highest price in this range is VAH (Value Area High) and the lowest is VAL (Value Area Low).
7. `drawValueArea()`: Draws the Value Area as a rectangle on the chart and can also draw separate VAH and VAL lines.

- POC Line Extension: If a new bar forms on the current chart, it extends the POC lines of recent profiles further to the right.
- `drawInfo()`: Displays informational text on the chart (e.g., "TPO Profile", "DayStartHour").

    - No `Sleep()`: Crucially, the `Sleep(5000);` call that was previously at the end of `start()` (causing freezes) has been removed, allowing the indicator to process efficiently.

4. Key Data Structures and Concepts

- `aprice_step[][3]` array: A 2D dynamic array that forms the backbone of each profile. For each profile calculated:
    - `aprice_step[index][PRICEIDX (0)]`: Stores the actual price value of a specific level.
    - `aprice_step[index][TPOIDX (1)]`: Stores the TPO count for that price level.
    - `aprice_step[index][VOLIDX (2)]`: Stores the accumulated volume for that price level.
- Profile Periods: Defined by `ProfileTimeframe`. The indicator determines the start and end times/bars for these periods (e.g., a "D" profile runs from 00:00 to 23:59, or a custom `DayStartHour`).
- `giDataTf` vs. `giProfileTf`:
    - `giProfileTf`: The timeframe that a single profile represents (e.g., `PERIOD_D1` for a daily profile).
    - `giDataTf`: The timeframe of the underlying bars used to build the profile (e.g., `PERIOD_M15` data might be used to construct a `PERIOD_H1` profile). This allows for finer granularity in profile construction.
- TPO vs. Volume Profile:
    - TPO (Time Price Opportunity): Each time a price level is traded within a segment of the profile period (typically one bar of `giDataTf`), the TPO count for that price level is incremented. It shows where price spent time.
    - Volume Profile: Shows how much volume was traded at each price level.
- POC (Point of Control): The price level within the profile with the highest TPO count (if TPO profile) or highest volume (if volume profile). This is considered a very significant level.
- VA (Value Area): The price range where a specified percentage (e.g., 70% via `VATPOPercent`) of the TPOs or volume occurred, centered around the POC. It represents an area of high acceptance.

5. Helper Functions

- `newBar()`: Detects if a new bar has started on the current chart's timeframe.
- Object Creation Functions (`createRect`, `createArw`, `createText`, `createTl`): Wrapper

functions to simplify the creation and updating of chart objects (rectangles for histogram/VA, arrows for open/close, text for labels, trendlines for POC/VAH/VAL). They handle object naming with `gsPref` and ensure objects are updated if they already exist.

- `delObjs()`: Deletes all chart objects created by this indicator that match the `gsPref` (and an optional sub-filter), used for cleanup.
- `RGB()`, `intToRGB()`: Utility functions for converting between RGB color components and MQL4's integer color representation.
- `addStr()`: A utility to pad strings with a character to a certain length.

6. Performance and Fixes

The code has undergone significant refactoring:

- The `Sleep()` call in `start()` was removed, which was the primary cause of the indicator freezing.
- Variables are now initialized upon declaration.
- Extensive comments have been added.
- Compiler warnings related to type conversions and undeclared identifiers (`CLR_NONE`, `-1` for `OBJ_ALL`) have been addressed to improve code robustness and correctness.

In essence, the indicator meticulously dissects price action over defined periods, aggregates TPO or volume data at discrete price levels, and then visually presents this information through histograms, POC lines, and Value Areas to aid in market analysis.