

Looking for MQL5 Coder to Create Trading Tool Library with Visual and ODO Meter Elements

I am looking for an experienced MQL5 coder to create a flexible library for my trading tool. The library will allow for the management of various trade objects (such as buy/sell boxes) and a separate ODO meter to visualize data. Both components need to be dynamic, customizable, and event-driven.

The library should contains :

Key Features Required:

1. Trading boxes, lines;

Properties:

openprice: Opening price for trades.

takeprofit[]: Support for multiple take-profit levels.

stoploss[]: Manage stop-loss levels.

Color: Customizable colors for the open price, take-profit, stop-loss, etc.

Transparency: Adjustable transparency for the boxes.

BMP Integration: Ability to add BMP files (such as arrows, icons) to the trading boxes.

Location and Movement: Dynamic placement and movement of boxes on the chart.

Methods:

create: Create a new box object for a trade.

addTakeProfit: Add multiple take-profit levels to the trade.

setTakeProfit: Modify an existing take-profit level.

addBMP: Attach BMP icons or arrows to the trading box for visualization.

setTransparency: Set the transparency level for the box and its components.

move: Relocate the box dynamically based on new market data.

Event Handling:

Event-driven updates using OnTick, OnTrade, or other custom triggers for real-time adjustments.

EXAMPLE Functions/Methods calling

```
boxobj boxx;
```

```
boxx.create("BuyBox", Buy, openprice);
```

```
boxx.addTakeProfit(tp1, price1);
```

```
boxx.addTakeProfit(tp2, price2);
```

```
boxx.setTakeProfit(tp1, newPrice1);
```

```
boxx.setTransparency(0.6); // Set 60% transparency
```

```
boxx.addBMP("ArrowUp", locationX, locationY); // Attach BMP icon
```

```
boxx.move(newLocationX, newLocationY); // Move box to a new position
```

Example Output:





2. ODO Meter

The ODO meter is a standalone object that serves to visualize data in real-time, with customizable levels, scales, and visual elements. It is separate from the trading boxes but will also support graphical elements.

Properties:

Levels: Define the levels of data representation on the meter.

Scale: Set the scale and intervals for the data.

Colors: Customizable color scheme for different levels on the ODO meter.

BMP Integration: Ability to add BMP files/icons for data visualization.

Transparency: Adjustable transparency for the ODO meter.

Location and Movement: Control the positioning and movement of the ODO meter.

Methods:

createODO: Create a new ODO meter object.

setLevel: Define levels, assign colors, and optionally attach BMP icons.

setScale: Set the data scale (for example, min/max values or increments).

setTransparency: Adjust transparency for different visual elements of the ODO meter.

move: Dynamically relocate the ODO meter on the chart.

Event Handling:

Handle events like data updates and market triggers to update the ODO meter dynamically using OnTick, OnTrade, etc.

Example Function/Method calling

odom obj;

```
obj.createODO("SpeedMeter", level1, level2, scale);
```

```
obj.setLevel(level1, color1, "SpeedIcon.bmp"); // Attach BMP icon at specific level
```

```
obj.setLevel(level2, color2); // Define additional level with different color
```

```
obj.setScale(0, 100); // Set the scale from 0 to 100
```

```
obj.setTransparency(0.5); // Set 50% transparency for the ODO meter  
obj.move(newLocationX, newLocationY); // Move ODO meter to a new position
```

Additional Considerations:

Both components (boxes and ODO meter) should be event-driven, optimized for real-time performance.

The library should allow dynamic and customizable visualizations.

The ODO meter is separate from the boxes but should have similar flexibility regarding transparency, movement, and BMP file integration.

Real-time data updates for the ODO meter should be handled smoothly through events (e.g., OnTick).

