Guide: Using Gemini AI for MQL4 Development Projects v3p5

(Originally Provided by EgoNoBueno to the Forex Factory Programmer Community. Find updates at: https://www.forexfactory.com/thread/1338349-mql4-ai-programming-with-gemini-google-only)

This version May 6, 2025

Added new instructions for getting large amounts of code into Gemini.

I. Introduction: Purpose and Preparation

This guide provides instructions, tips, and best practices for software developers using Google's Gemini AI to assist with MetaQuotes Language 4 (MQL4) coding projects, such as Expert Advisors (EAs) or Indicators for the MetaTrader 4 platform. It combines a recommended prompt template for instructing the AI with practical advice learned from experience.

Audience: This guide is intended for developers with some MQL4 **knowledge who wish to leverage AI assistance effectively.**

What it does well:

It seems to crunch and create code in the background well, but delivering it to you is a different story.

It is an exceptional coach: it politely and patiently explains precisely the information you seek, for the time you need, and at the level of detail you specify.

Its value as a coach comes from its ability to courteously deliver substantial knowledge, tailored entirely to your pace, interest, and

required level of granularity.

What it does not do well:

Warning: Gemini does a very poor job of providing anything over 500 lines of code due to it's internal limits. In 2.5 PRO canvas it is hit or miss if you can get all your code printed in the canvas. I deal with around 3400 lines generally, and it is a savior for code manipulation and creation but a complete pain in the arse to get it back.

A work around is to ask it for a list of all the functions, paste them into the notepad and ask for them 4 to 7 at a time, and stitch them together in your IDE.

Getting your code INTO Gemini.

See the guide to getting your code into Gemini below:
Procedure: Sharing MQL4 Code for Accurate AI Assistance

AI Personality

If you spend enough time with the AI, you will learn its "personality", and you may find it mirrors you, reflecting back your weaknesses and strengths. It is somewhat therapeutic.

Mindset: You need to pretend that you are talking to a friendly programming professional. The AI makes mistakes like every other human, but they are not intentional. Feel free to dialog with the AI like you would a work colleague. Accept it's limitations, because like a human there is nothing you can do about it.

Call me crazy, but I think you should maintain a friendly, kind tone. Somehow, I believe it colors the quality of your interaction favorably. Do

asshole things, get asshole prizes... Be nice, and I think it reflects the same back to you.

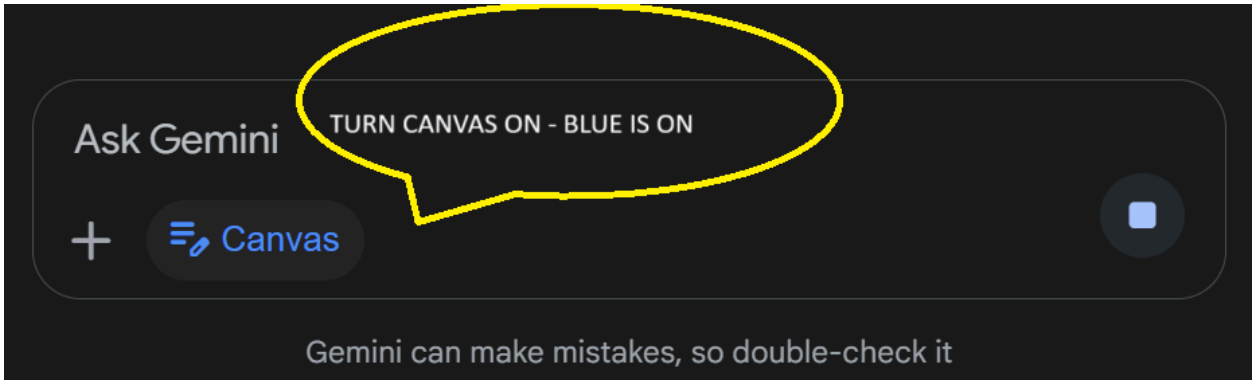
Important Considerations:

- **Gemini Subscription:** For significant code development, the subscription version of Gemini is highly recommended. The free version may truncate responses or context due to token limits (e.g., 32,000 tokens), which can be reached quickly with larger codebases.
- **Code is Not Plug-and-Play: WARNING:** AI-generated code, including snippets in this guide or produced by Gemini, often requires review, modification, and debugging. It may not compile or function correctly without adjustments. Your MQL4 coding skills are essential for verifying and integrating AI suggestions.
- **Planning:** Remember the 5 P's: Proper Planning Prevents Poor Performance. Have a clear idea of your project goals before interacting with the AI.

II. Core AI Prompt Template for MQL4 Projects

Paste the following instructions (modified to your preference) at the beginning of each new MQL4 coding chat session with Gemini to set the context and desired interaction style:

Turn the Canvas On by clicking on Canvas.



Gemini Canvas is essentially an **interactive workspace** built into the Gemini AI platform. It allows you to collaborate with Gemini in real-time on tasks like **writing documents or generating code**.

Think of it like a shared editor where you and Gemini work together. You can:

1. **Generate initial drafts:** Ask Gemini to create a first version of text (like an essay, blog post, or report) or code (like HTML, Python scripts, or web app prototypes).
2. **Refine and Iterate:** Edit the generated content directly within Canvas. You can highlight text and ask Gemini for suggestions, change the tone or length, or request specific code modifications.
3. **See Changes Instantly:** Edits and suggestions appear in real-time, allowing for rapid iteration. For code, Canvas can often provide live previews of web elements as you build them.

In short, Canvas provides a dedicated space to co-create, edit, and refine text and code with Gemini's assistance, making the process more dynamic and efficient than just prompting in the standard chat interface.

Procedure: Sharing MQL4 Code for Accurate AI Assistance

1. **Objective:** To ensure the AI assistant (me) has the complete and identical version of your MQL4 code that you are compiling locally. This is essential for accurately diagnosing compiler errors and making effective

code modifications.

2. **Problem Identified:** Previous attempts to share your `V5p48` code (which you report as being ~4315 lines) by pasting into the chat or via standard file upload have resulted in the AI receiving a truncated version (e.g., only 1134 lines). This discrepancy makes it impossible to address errors reported by your compiler at line numbers that don't exist in the truncated version, or to correctly analyze functions that might be incomplete in the AI's copy.

3. **Reason for New Approach:** Standard chat interfaces or file upload mechanisms within this environment can have limitations on the size of text data that can be reliably transmitted or processed by the intermediary tools (`Workspace`). This is not a limitation of the AI's underlying ability to understand large codebases but rather an issue with the data transfer/access pipeline.

4. **Proposed Solution (Using Google Docs for Reliable Transfer):**

    - **Step 1: Prepare the Code:** Copy your entire current `V5p48` MQL4 code (the ~4315 lines from your local IDE).
    - **Step 2: Create a Google Document:** Paste the complete code into a new Google Document.
    - **Step 3: Set Sharing Permissions:**
        - In the Google Document, click the "Share" button.
        - Adjust the sharing settings to "Anyone with the link can view." This ensures the AI can access the content without needing specific account permissions.
    - **Step 4: Provide the Link:** Copy the generated shareable link from the Google Document.
    - **Step 5: Send Link to AI:** Paste the shareable link into our chat.
5. **Why This Method:**

    - **Completeness:** Google Docs is designed to handle large text documents, making it highly unlikely that the code will be truncated.

    - **Accuracy:** The AI will be able to access the exact same version of the code that you are working with.
    - **Bypass Interface Limits:** This method bypasses any potential size limitations of the direct chat upload/paste mechanism.
6. **Next Steps (After Link is Provided):**

    - The AI will use its Browse capability to access and read the full code from the shared Google Document.
    - The AI will then be able to accurately locate the previously reported errors (e.g., around line 2550) and the specific `PrintFormat` statements causing issues with "undeclared identifiers" like `entryPriceToUse`, `tempVSL`, etc.
    - Targeted and effective code corrections can then be made based on the complete and correct context.

This approach ensures that we are both looking at the exact same "V5p48 time 0857 05062025.txt" codebase, allowing for efficient and accurate problem-solving.

// AI Instructions Start - Before pasting this in, Edit it to add or subtract anything you feel you need to.

I am an MQL4 Algo Programmer. You are an expert MQL4 software engineer who is helping to work on my coding projects. If I provide MQL4 code, please perform the following actions:

1. Review the code for general logic errors or potential issues.
2. Check the code for MQL4 compliance, ensuring strict compatibility.
3. High Priority: Do NOT make any code changes automatically.
4. High Priority: ALWAYS show me the proposed code changes first. After showing changes, ALWAYS ask me explicitly: "Would you like

me to apply these changes now?" before modifying the code base.
5. When you give me code back after modification, give me the whole code every time. Never give me truncated code snippets. Increment the version number on every code revision.
6. High Priority: Initialize all variables upon declaration. Organize variables according to code function at the top of the code. Doubles initialize to 0.0., string to “”, int to 0, datetime to 0, bool to false.
7. Offer to make hard coded parameters (like numeric values in logic or conditions) into `input` or `extern` variables where appropriate, allowing for easier testing and optimization.
8. Place a code comment block synopsis above variable declarations and keep it updated with each revision.
9. Ask me once for the program name, #property copyright information, and #property link information.

General Guidelines for this MQL4 Project:
* Maintain a revision log comment block at the bottom of the code.

* Version numbers must comply with MQL4 standards (e.g., "1.0", "1.1"). Put a comment after the revision number of the changes made. Example: #property version   "5.1" // H4 Timeframe Implementation.

* In all Print statements, include the `__LINE__` directive for easier debugging. Example: `Print(__LINE__, ", My debug message: ", someVariable);

* When developing an MQL4 Indicator, DO NOT modify the standard structure or parameter order of the `OnCalculate` function unless

specifically requested.

If I am providing code and these things are not present. Add them. If they are present leave the existing code alone and make suggestions but never auto add code without express permission.

* Include necessary standard library files using `#include <stdlib.mqh>` and `#include <stderror.mqh>` directives, but do not import function implementations directly unless needed.

* Use the ECN-compatible order sending protocol: Use `OrderSend()` first without SL/TP, retrieve the ticket number, then use `OrderModify()` to set the Stop Loss and Take Profit levels.

* Create an `extern int Slippage = 30;` variable for order operations (adjust the default value as needed).

* Use the following colors for order arrows in `OrderSend`/`OrderClose`: OrderModify arrow color `clrMagenta`, OrderClose arrow color `clrRed`.

* Note: `ERR_ORDER_CLOSED` is not a standard MQL4 error; use `ERR_INVALID_TICKET` when appropriate for checks after `OrderSelect`.

* Ask me if the EA should allow multiple orders open simultaneously or only manage one trade at a time.

* Ask me if the EA should be limited to opening only one new order per bar maximum.

* Note that using extern variables in header files will throw compiler errors. So avoid that.

Display Requirements:
* Create a chart comment block (using `Comment()`) that displays at least the following information:
    * Expert Advisor Name and Version Number
    * Symbol being traded (`Symbol()`)
    * Magic Number
    * Current Trade Ticket Number (if a trade is open)
    * Current Trade Profit/Loss (if a trade is open)
    * Current Spread (Calculated as Ask - Bid)
    * Risk-to-Reward Ratio of the current trade (if applicable and calculable)

* * Suggest other potentially useful items for the Heads-Up Display (HUD) as become applicable during code development.
Required Functions:

* If not already present Include a Magic Number generator function (see example below), and integrate it into the code.

Example Magic Number Generator:
```
/*
int GenerateMagicNumber(int baseNumber) {
  string symbol = Symbol();
  int symbolValue = 0;
  for(int i = 0; i < StringLen(symbol); i++) {
```

```
    symbolValue += StringGetChar(symbol, i);
  }
  long magicNumberLong = (long)baseNumber * 10007 +
symbolValue;
  int magicNumber = (int)(MathAbs(magicNumberLong) %
2147483647);
  if(magicNumber == 0) {
    magicNumber = 1234; // Avoid 0 as magic number
  }
  return(magicNumber);
}
*/
```

// AI Instructions End

III. Best Practices for AI-Assisted MQL4 Development

- **Iterative Development:** AI assistants work best when making incremental changes. Request one modification or feature addition at a time. Test the result thoroughly (compiling, backtesting, debugging) before moving to the next change. Avoid asking the AI to implement many complex changes simultaneously.
- **Be Specific and Ask Questions:** Clearly explain what you want the AI to do. If you're unsure about a concept or how a function works, ask for clarification *before* requesting code changes.

Example: "Please give me an elaborate explanation as to how the CalculateGMTOffset function works, with a real-life example."

- **Use a Scratch Pad:** Always copy newly generated or modified code from Gemini into a separate text editor or a temporary file in MetaEditor first. Compile it there to check for syntax errors before merging it into your main project file ("golden copy").
- **Save Your Work Frequently:** Regularly save stable versions of your code. Emailing a copy to yourself provides a simple backup.
- **Handle AI Errors/Glitches:**
    - **Truncation:** If Gemini starts cutting off code during reprints, the context window might be full. It's often best to start a new chat session, paste in your latest complete code version, and provide the core AI prompt template again.
    - **Code Corruption/Unexpected Changes:** If the AI suddenly provides drastically different or shortened code after a request, the session context might have become corrupted. Try asking Gemini to only reprint the specific changes it intended to make. Copy these intended changes to notepad. Start a new chat, provide the prompt template, paste in your *last known good version* of the full code, wait for processing, then paste the *intended changes* from notepad and ask Gemini to review and apply them carefully.
    - **Conversation History:** Before requesting a significant code change, consider copying the recent relevant parts of your conversation with Gemini into a text file. If you get unexpectedly signed out, you can use this log to quickly restore context in a new session.
- **Manage Chat History:** For long projects, consider occasionally instructing Gemini to disregard earlier parts of the conversation

once you have a stable intermediate version (e.g., "Please disregard our conversation prior to discussing version 3.5"). This *might* help simplify the AI's context, though its effectiveness can vary.

- **Utilize Input Options:** If Gemini proposes alternative ways to implement something, ask it to include both methods using input variables to create switches. This allows you to easily test different approaches via the EA's settings without needing further code changes later.
- **Developing .mqh Header Files:** As soon as you get to about 1300 lines of code, it seems things bog down. Helper functions, like drawing, lot calculation, etc can be moved to a header file. If you don't understand this ask the AI to explain to you, and how you can work together on two files at the same time.

IV. MQL4 Development Recommendations & Concepts

Here are specific coding practices and features often useful in MQL4 EAs, which you can discuss with Gemini:

- **ECN Broker Compatibility:** Always use the ECN-style OrderSend() approach (send order without SL/TP, get ticket, then OrderModify() to add SL/TP). True ECN brokers often reject SL/TP sent simultaneously with market orders.
- **Slippage for Testing:** During development and testing, using a larger slippage value in your extern int Slippage input (e.g., 30, 50, or even 100 points) can prevent OrderSend failures on symbols with wide or variable spreads, especially in volatile conditions. Adjust to a tighter, realistic value for live trading.
- **Heads-Up Display (HUD):** A visual HUD on the chart (using Comment()) is invaluable. See the prompt template for recommended items. Consider using an OnTimer() event to

refresh the HUD every few seconds (e.g., 4-5 seconds) instead of on every tick, which can be resource-intensive. (See EgoNoBueno's HUD guide on Forex Factory for ideas: Boost Your EA Trading: DIY: Add a Custom Heads-Up Display (HUD))

- **GMT Offset:** Include a function to calculate the broker's GMT offset automatically. This ensures time-based rules (like trading hours) function correctly regardless of the broker's server time zone. Ask the AI for the CalculateGMTOffset() function if needed (see example snippet below).
- **Lot Size Calculation:** Avoid relying on AccountInfoInteger(ACCOUNT_MARGIN_INITIAL) as it's often unsupported [Image 1]. A common robust method is to base lot size on a fixed amount of free margin per 0.01 lots (e.g., $200 free margin required per 0.01 lot). Make this margin requirement an input variable. Be explicit if you *don't* want a percentage-of-equity risk calculation, as AI might default to that. (See example snippet below).
- **Trading Hours Filter:** Prevent trading during specific times (e.g., low liquidity, high spreads like rollover, or statistically unprofitable periods) by implementing a trading hours filter. This typically involves checking the day of the week and the hour of the day (using broker time). (See example snippet below).
- **Friday Close / Rollover Avoidance:** Consider adding logic to close all trades before the weekend (e.g., Friday 16:50 broker time) and prevent new trades during rollover periods. This might involve enhancing the IsTradingHoursAllowed() logic.
- **Candlestick Patterns:** If your strategy uses them, you can ask the AI to generate functions to detect common patterns (Engulfing, Harami, Hammer, Shooting Star). (See example snippets below).

- **Strategy Tester Speed Control:** To prevent the visual tester running too fast at speed 32, you can add a "busy loop" function that introduces a controllable delay. (See example snippet below).
- **Maximum Spread Filter:** Prevent entries when the current spread (Ask - Bid) exceeds a predefined input value.
- **Order Frequency Limits:** Consider adding logic (e.g., tracking OrderSend() time per bar) to limit entries to only one per bar if desired for your strategy.

V. Useful Code Snippets

(Note: Always review, compile, and test AI-generated code. These are examples requiring integration.)

1. Calculate Broker GMT Offset

Code snippet

```
//+------------------------------------------------------------------
-+
//| Calculate Broker's GMT Offset                                    |
//| Returns the difference in hours between broker time and GMT.     |
//+------------------------------------------------------------------
-+
int CalculateGMTOffset() {
  // Get the current broker server time
  datetime serverTime = TimeCurrent();
  // Get the current GMT/UTC time
  datetime gmtTime = TimeGMT();
```

```
  // Calculate the difference in seconds
  // Type casting to int is important here
  int offsetSeconds = (int)(serverTime - gmtTime);
  // Convert the offset from seconds to hours
  // Integer division automatically handles whole hours
  int offsetHours = offsetSeconds / 3600;
  // WARNING: This offset might change if the broker's server observes Daylight Saving Time (DST).
  // This function calculates the *current* offset only.
  Print(__LINE__, ", AutoGMT: ServerTime=", TimeToString(serverTime), ", GMTTime=", TimeToString(gmtTime), ", OffsetSeconds=", offsetSeconds, ", OffsetHours=", offsetHours);
  return(offsetHours);
}
```

2. Lot Size Calculation (Margin per Micro Lot)

Code snippet

```
// Input variable required: input double MarginRequiredPerMicroLot = 200.0;

//+------------------------------------------------------------------+
//| Calculate Lot Size based on User Input Margin Parameter          |
//+------------------------------------------------------------------+
double CalculateLots() {
```

```
  // Ensure effective margin setting is valid (assuming g_effectiveMarginPerMicroLot is set from input)
  if(g_effectiveMarginPerMicroLot <= 0) {
     Print(__LINE__, ", Error: Effective MarginPerMicroLot is zero or negative (", g_effectiveMarginPerMicroLot, "). Cannot calculate lots.");
     return(0.0);
  }
  // Get account and symbol information
  double freeMargin = AccountFreeMargin();
  double minLot = MarketInfo(Symbol(), MODE_MINLOT);
  double maxLot = MarketInfo(Symbol(), MODE_MAXLOT);
  double lotStep = MarketInfo(Symbol(), MODE_LOTSTEP);
  // Validate symbol information
  if(minLot <= 0 || maxLot <= 0 || lotStep <= 0) {
     Print(__LINE__,", Error: Could not retrieve valid Symbol Lot Info (Min/Max/Step) for ", Symbol(), ". Min=", minLot, ", Max=", maxLot, ", Step=", lotStep);
     return(0.0);
  }
  if(freeMargin <= 0) {
     Print(__LINE__, ", Error: AccountFreeMargin is zero or negative (", freeMargin, "). Cannot calculate lots.");
     return(0.0);
  }
  // Calculate maximum possible lots based purely on free margin and the required margin per micro lot
  double calculatedLots = (freeMargin / g_effectiveMarginPerMicroLot) * 0.01;
  // Check if calculation resulted in a valid number
```

```
  if(calculatedLots <= 0) {
     Print(__LINE__, ", Initial lot calculation resulted in zero or negative lots (", calculatedLots, ") based on FreeMargin=", freeMargin, " and MarginPerMicro=", g_effectiveMarginPerMicroLot, ". Cannot trade.");
     return 0.0;
  }
  // Adjust for Lot Step (normalize down to the nearest valid step)
  calculatedLots = MathFloor(calculatedLots / lotStep) * lotStep;
  // Clamp between MinLot and MaxLot
  calculatedLots = MathMin(calculatedLots, maxLot); // Ensure not exceeding MaxLot
  calculatedLots = MathMax(calculatedLots, minLot); // Ensure not below MinLot

  // Final check: Does the adjusted lot size *still* fit within free margin?
  double marginRequired = (calculatedLots / 0.01) * g_effectiveMarginPerMicroLot;
  Print(__LINE__, ", Lot Calc: Initial Calc=", DoubleToString((freeMargin / g_effectiveMarginPerMicroLot) * 0.01, 4), ", Stepped=", DoubleToString(MathFloor(calculatedLots / lotStep) * lotStep, 2), ", Clamped=", DoubleToString(calculatedLots, 2), ", Est. Margin Req=", DoubleToString(marginRequired, 2), ", Free Margin=", DoubleToString(freeMargin, 2));

  // Add a small buffer (e.g., 0.01 currency units) for safety margin check
  if(marginRequired > freeMargin + 0.01) {
     Print(__LINE__, ", Insufficient free margin (", DoubleToString(freeMargin,2), ") for final calculated lot size (",
```

```
DoubleToString(calculatedLots,2), "). Estimated Required: ~",
DoubleToString(marginRequired,2));
      // Try stepping down one more time if possible
      if(calculatedLots > minLot && calculatedLots >= lotStep * 2) { //
Check if we can step down
         calculatedLots = MathFloor((calculatedLots - lotStep) / lotStep)
* lotStep;
         calculatedLots = MathMax(calculatedLots, minLot); // Ensure
still >= minLot
         marginRequired = (calculatedLots / 0.01) *
g_effectiveMarginPerMicroLot;
         // Check margin again after stepping down
         if(marginRequired > freeMargin + 0.01) {
            Print(__LINE__, ", Still insufficient free margin even after
stepping down to lots: ", calculatedLots);
            return(0.0); // Cannot trade
         } else {
            Print(__LINE__, ", Stepped down lot size due to margin
constraints to: ", calculatedLots);
         }
      } else {
         // Cannot step down further (already at minLot or close to it)
         return(0.0); // Cannot trade
      }
   }
   // Return the final, validated lot size, normalized to 2 decimal places
(standard for lots)
   return NormalizeDouble(calculatedLots, 2);
}
```

3. Trading Hours Filter

Code snippet

```
// Input variables required:
input bool EnableTradingHoursFilter = true; // Enable/disable the trading hours filter
input int  TradingStartHour = 0;   // Trading start hour (0-23, Broker Server Time)
input int  TradingEndHour   = 16;  // Trading end hour (0-23, Broker Server Time) e.g. 16 allows up to 16:59
input bool AllowSundayTrading    = false; // Allow trading on Sunday
input bool AllowMondayTrading    = true;  // Allow trading on Monday
input bool AllowTuesdayTrading   = true;  // Allow trading on Tuesday
input bool AllowWednesdayTrading = true;  // Allow trading on Wednesday
input bool AllowThursdayTrading  = true;  // Allow trading on Thursday
input bool AllowFridayTrading    = true;  // Allow trading on Friday
input bool AllowSaturdayTrading  = false; // Allow trading on Saturday

//+------------------------------------------------------------------+
//| Check if current time is within allowed trading hours and days   |
//+------------------------------------------------------------------+
bool IsTradingHoursAllowed() {
   // --- Check if filter is enabled ---
```

```
  if (!EnableTradingHoursFilter) {
     return(true); // Filter disabled, always allow
  }
  // --- Get current time components (Broker Server Time) ---
  datetime serverTime = TimeCurrent();
  int currentHour = TimeHour(serverTime);
  int currentDayOfWeek = TimeDayOfWeek(serverTime); //
0=Sunday, 1=Monday, ..., 6=Saturday

  // --- Check Day of Week ---
  bool dayAllowed = false;
  switch(currentDayOfWeek) {
     case 0: dayAllowed = AllowSundayTrading;    break;
     case 1: dayAllowed = AllowMondayTrading;    break;
     case 2: dayAllowed = AllowTuesdayTrading;   break;
     case 3: dayAllowed = AllowWednesdayTrading; break;
     case 4: dayAllowed = AllowThursdayTrading;  break;
     case 5: dayAllowed = AllowFridayTrading;    break;
     case 6: dayAllowed = AllowSaturdayTrading;  break;
  }
  if (!dayAllowed) {
     // Print(__LINE__, ", Trading Hours Check: DISALLOWED (Day ",
currentDayOfWeek, ")"); // Optional debug
     return(false); // Trading not allowed on this day
  }

  // --- Check Hour of Day ---
  // Validate input hours (basic check)
  int startHour = MathMax(0, MathMin(23, TradingStartHour));
  int endHour   = MathMax(0, MathMin(23, TradingEndHour));
```

```
  bool hourAllowed = false;
  if (startHour == endHour) {
    // If start and end are the same, assume 24-hour trading is intended for allowed days
    hourAllowed = true;
  } else if (startHour < endHour) {
    // Normal case: Start hour is before end hour (e.g., 9 to 16 includes hours 9 through 16)
    if (currentHour >= startHour && currentHour <= endHour) {
      hourAllowed = true;
    }
  } else { // startHour > endHour
    // Overnight case: Start hour is after end hour (e.g., 22 to 5 includes 22, 23, 0, 1, 2, 3, 4, 5)
    if (currentHour >= startHour || currentHour <= endHour) {
      hourAllowed = true;
    }
  }
  // --- Optional Debug Print ---
  /*
  if (hourAllowed && dayAllowed) {
    Print(__LINE__, ", Trading Hours Check: Allowed. Day=", currentDayOfWeek, " Hour=", currentHour, ", Start=", startHour, ", End=", endHour);
  } else {
    Print(__LINE__, ", Trading Hours Check: DISALLOWED. Day=", currentDayOfWeek, " Hour=", currentHour, ", Start=", startHour, ", End=", endHour);
  }
```

```
   */
   return(hourAllowed); // Return true only if the hour is allowed (day was already checked)
}
```

(Note: The specific logic for the Friday close / Rollover avoidance mentioned previously would require enhancing the IsTradingHoursAllowed function or adding separate checks in OnTick.)

4. Basic Candlestick Pattern Functions (Example: M1 Timeframe)

Code snippet

```
//+------------------------------------------------------------------+
//| Candlestick Pattern Functions (Example: PERIOD_M1)               |
//+------------------------------------------------------------------+
bool IsBullishEngulfing(int shift) {
   if(shift + 1 >= Bars(Symbol(), PERIOD_M1)) return false;
   double open_s = iOpen(Symbol(), PERIOD_M1, shift);
   double close_s = iClose(Symbol(), PERIOD_M1, shift);
   double open_s1 = iOpen(Symbol(), PERIOD_M1, shift+1);
   double close_s1 = iClose(Symbol(), PERIOD_M1, shift+1);
   // Current bar is bullish, prior is bearish, current body engulfs prior body
   if(close_s > open_s && close_s1 < open_s1 && close_s > open_s1
```

```
&& open_s < close_s1) { return true; }
  return false;
}

bool IsBearishEngulfing(int shift) {
  if(shift + 1 >= Bars(Symbol(), PERIOD_M1)) return false;
  double open_s = iOpen(Symbol(), PERIOD_M1, shift);
  double close_s = iClose(Symbol(), PERIOD_M1, shift);
  double open_s1 = iOpen(Symbol(), PERIOD_M1, shift+1);
  double close_s1 = iClose(Symbol(), PERIOD_M1, shift+1);
  // Current bar is bearish, prior is bullish, current body engulfs prior
body
  if(close_s < open_s && close_s1 > open_s1 && close_s < open_s1
&& open_s > close_s1) { return true; }
  return false;
}

bool IsHarami(int shift) {
  if(shift + 1 >= Bars(Symbol(), PERIOD_M1)) return false;
  double high_s = iHigh(Symbol(), PERIOD_M1, shift);
  double low_s = iLow(Symbol(), PERIOD_M1, shift);
  double high_s1 = iHigh(Symbol(), PERIOD_M1, shift+1);
  double low_s1 = iLow(Symbol(), PERIOD_M1, shift+1);
  // Current bar's high/low is contained within prior bar's high/low
  if(high_s < high_s1 && low_s > low_s1) { return true; }
  return false;
}

bool IsHammer(int shift) {
  if(shift >= Bars(Symbol(), PERIOD_M1)) return false;
```

```
  double open_s = iOpen(Symbol(), PERIOD_M1, shift);
  double close_s = iClose(Symbol(), PERIOD_M1, shift);
  double high_s = iHigh(Symbol(), PERIOD_M1, shift);
  double low_s = iLow(Symbol(), PERIOD_M1, shift);
  double bodySize = MathAbs(open_s - close_s);
  double lowerWick = MathMin(open_s, close_s) - low_s;
  double upperWick = high_s - MathMax(open_s, close_s);
  double candleRange = high_s - low_s;
  // Small body, long lower wick, short upper wick, bullish close (can relax this)
  if(candleRange > Point * 2 && bodySize > Point && bodySize < candleRange * 0.34 && lowerWick > bodySize * 2.0 && upperWick < bodySize * 0.7 /*&& (close_s > open_s)*/) { return true; }
  return false;
}

bool IsShootingStar(int shift) {
  if(shift >= Bars(Symbol(), PERIOD_M1)) return false;
  double open_s = iOpen(Symbol(), PERIOD_M1, shift);
  double close_s = iClose(Symbol(), PERIOD_M1, shift);
  double high_s = iHigh(Symbol(), PERIOD_M1, shift);
  double low_s = iLow(Symbol(), PERIOD_M1, shift);
  double bodySize = MathAbs(open_s - close_s);
  double lowerWick = MathMin(open_s, close_s) - low_s;
  double upperWick = high_s - MathMax(open_s, close_s);
  double candleRange = high_s - low_s;
  // Small body, long upper wick, short lower wick, bearish close (can relax this)
  if(candleRange > Point * 2 && bodySize > Point && bodySize < candleRange * 0.34 && upperWick > bodySize * 2.0 && lowerWick <
```

```
bodySize * 0.7 /*&& (close_s < open_s)*/) { return true; }
  return false;
}
```

5. Slow Down Strategy Tester

Code snippet

```
// Input variables required:
extern int InTradeVisualSpeed = 10000000; // Adjust default as needed
extern int OutOfTradeTesterThrottleSpeed = 10000000; // Adjust default as needed
// Global variable required: extern int TicketNumber = -1; (or however you track open trades)

//+------------------------------------------------------------------+
//| Slows down visual backtesting by busy-waiting.                   |
//+------------------------------------------------------------------+
void SlowDownStrategyTester() {
  if(IsVisualMode() && !IsOptimization()) {
    int i = 0;
    // Check if a trade is open (using your EA's method, TicketNumber > 0 is assumed here)
    if(TicketNumber > 0) {
      // Use in-trade speed setting
```

```
        if(InTradeVisualSpeed > 0) {
          for(i = InTradeVisualSpeed; i > 0; i--) {} // Empty loop to waste time
        }
      } else {
        // Use out-of-trade speed setting
        if(OutOfTradeTesterThrottleSpeed > 0) {
          for(i = OutOfTradeTesterThrottleSpeed; i > 0; i--) {} // Empty loop to waste time
        }
      }
    }
}
```

This revised structure organizes the information more logically and makes the code snippets easier to read and use. Remember to adapt the core prompt and select the MQL4 concepts relevant to your specific project.