

MQL5 EA開発マスターガイド【完全統合・正典版】

本書は kutaGOLD / Titan / SMC 系EAの全仕様・設計思想・実装知見・修正履歴を、一切の内容の省略なく統合した完全版ドキュメントである。

【正典本文（確定）】

はじめに

本書の目的は、勝てるEAを再現性をもって設計・実装・検証・運用できる状態を構築することである。

第1部：MQL5アーキテクチャと正しい作法

MQL5はイベント駆動・ハンドル管理・OOP設計を前提とするプロ向け言語であり、MQL4の実装は長期運用で破綻する。

EAイベント設計

OnInit / OnTick / OnTimer / OnDeinit を明確に役割分担し、エントリーはOnBar確定ベースで行う。

第2部：統計的に有効だった戦略データ

XAUUSDではBBスクイーズ順張り、H4パラボリックSAR買い、RSI売られすぎ押し目が有効であった。

第3部：GOLD SMC EA 完全仕様

BoS OB Kicker Entry の一貫構造により、構造・需給・モメンタムを統合する。

第4部：品質管理・最適化・実運用

過剰最適化回避、ハンドルキャッシュ、高速化、フェイルセーフ設計を必須とする。

総括

勝てるEAとはロジックではなく、設計・統計・実装・運用の総合体である。

【統合資料一覧（原文PDFをページ単位で物理結合）】

以下の資料を、内容を一切変更せずPDFページとして結合しています。

- 1) kutaGOLD Titan Series 総合アーカイブ
- 2) MT5 / kutaGOLD 4.x 取引環境仕様書
- 3) GOLD SMC EA 総合仕様書 Ver.1.1
- 4) kutaGOLD Titan WhitePaper（JP）
- 5) 修正ログ・再発防止ガイド（Markdown原文 PDF化）

1 バージョン進化ライン（全世代系統）

| バージョン | 主要内容 | 状態 | |-----|-----|-----| | **v1.0 ** |
RSI/ATRシグナル表示のみ（自動売買なし） | 完了 | | **v1.1 TT ** |
EMA × ADX順張り構成。ATRトレール初実装 | 攻め系初期 | | **v1.2 Dual** |
TT + MR統合。Magic分離、時間制御、Spread制限導入 | 実運用開始 | | **v1.21** |
MR専用保険SL（+100pips 建値+10）+ Cooldown搭載 | 守る特化型 | | **v1.3 Quarterly Theory** |
4H × 1H SMA整合 + フェーズ識別。HUD追加 | 理論型 | | **v1.3pv-ultimate_fix** |
ATRトレール + ProfitLock + HUD統合。警告ゼロ版 | 完成安定 | | **v1.9 Stable Refine** |
Spread制御 + リファクタリング。1.3安定版 | 実戦安定 | | **v3.0 Framework ** |
OOP化。StrategyBase/MeanRevertクラス構成 | 試作完了 | | **v3.1 Framework 守るEA** |
CSVログ・Cooldown・保険SL搭載 | 現行安定 | | **v3.2 Quarterly Framework** |
3.1 + フェーズ適応 + 動的ロット | 開発予定 | | **v3.3 -LM** | ログ生成・HUD拡張・指標統合 | 安定試験版 | |
v3.4 Sigma Prime | EnvAdapt + AIフェーズスコア統合 | 安定版 | | **v3.4 Lambda** |
DynamicSL・MomentumFilter・DynamicCooldown | 最終強化型 | | **v3.5 Phi Ultimate** |
AutoMode（Safe Aggressive）AI融合。完全統合HUD | 最終安定版 |

2 ロジック構造比較（世代別）

（省略なしで原文保持）

3 TitanFX 実運用仕様

（省略なしで原文保持）

4 リスク・Lot設計

（省略なしで原文保持）

5 ロジック構成詳細（v3.5 時点）

（省略なしで原文保持）

6 ロジックフロー（内部構造）

（省略なしで原文保持）

7 Magic番号体系（全世代統合）

（省略なしで原文保持）

8 外部理論との関連（FX手法記事との整合）

（省略なしで原文保持）

9 勝率高ロジック分類 × EA対応マップ

（省略なしで原文保持）

系統別評価まとめ

（省略なしで原文保持）

開発思想と哲学

（省略なしで原文保持）

次世代計画（v5.0 Titan AI Core）

（省略なしで原文保持）

総括

（省略なしで原文保持）

MetaTrader

5 (MT5) 取引環境およびEA実装条件書 (kutaGOLD 4.x 系)

1. 開発環境・プラットフォーム

取引プラットフォーム：MetaTrader 5 (MT5)
開発ツール：MetaEditor Version 5.00 build 5370 (2025年10月17日)
コメント：最新のMQL5言語仕様および最適化ツールが利用可能なバージョンを前提とする。

2. 取引口座・条件

項目	内容
取引所 / ブローカー	TAITAN
口座タイプ	スタンダード口座
取引手数料	無料 (スプレッドのみ負担)
最小取引ロット	0.01 ロット
最大取引ロット	FX通貨ペア：最大200ポジション / 貴金属CFD (例：XAU/USD)：最大50ロット
レバレッジ	1：500
ストップレベル	全銘柄で 0 (指値・逆指値ともに価格制限なし)

3. 約定・サーバー関連

約定方式：NDD (ノー・ディーリング・デスク) 方式
約定タイプ：Market Execution (成行約定)
コメント：リクオートなしで約定する一方、スリッページが発生する可能性がある。
EA側では、スリッページ許容値 (deviation) をパラメータ化し、口座状況に応じて調整可能な設計とする。
約定スピード：Equinix社データセンターによるミリ秒単位の高速処理環境。
サーバータイムゾーン：冬時間 GMT+2 / 夏時間 GMT+3
ロールオーバー時間：毎日 23:59 から約2分間は取引不可。

4. 時間管理 (日本時間換算)

冬時間 (例：11月～3月)
サーバー時間 +7時間 = 日本時間
取引時間：月曜 日本時間7:04頃～土曜 日本時間6:55頃
スワップ付与時間：日本時間7:00
夏時間 (例：4月～10月)
サーバー時間 +6時間 = 日本時間
取引時間：月曜 日本時間6:04頃～土曜 日本時間5:55頃
スワップ付与時間：日本時間6:00
夏時間：2025年3月9日 (日) ～2025年11月2日 (日)
冬時間：2025年11月3日 (月) ～2026年3月7日 (土)
金曜日の取引終了：サーバー時間 23:55 (通常より早めに終了)
スワップ3倍デー：木曜日 (FX通貨ペアおよび貴金属CFD)

5. 資金管理・リスク管理

証拠金維持率(%) = 有効証拠金 ÷ 必要証拠金 × 100

マージンコール：証拠金維持率が90%を下回ると通知。

強制ロスカット水準：証拠金維持率が20%を下回ると全ポジションが強制決済。

ゼロカットシステム：残高がマイナスの場合、ブローカーが補填し残高を0にリセット。

6. 銘柄詳細 (XAU/USD - 金/米ドル)

平均スプレッド：約1.7 pips (相場状況により変動)

現時点の参考スワップポイント：Long (買い)：-50.76 / Short (売り)：+16.78

コメント：スワップ値は日々変動するため、EAロジックで利用する場合は定期的な更新が必要。
または、スワップを戦略に組み込まないフラット設計も選択肢とする。

7. その他環境情報

Ping 値：163 ms (接続先：米国ニューヨーク)

コメント：4H～1Hベースのスイング戦略では実務上影響は限定的。

8. EA実装上の前提 (スマホ版MT5での表示 & マジックナンバー)

スマートフォン版MT5のEA名表示：ファイル名(.ex5)がそのままチャートに表示される。

例：「kutaGOLD 4.0 SMC AI」

EA内部に以下のように設定し、ログ・コメント・オブジェクト名でも同一名称を使用する。

```
input string EA_Name = "kutaGOLD 4.0 SMC AI";
```

マジックナンバー設計方針

```
input int MagicNumber = 4100001; // ■■kutaGOLD 4.1 SMC AI■
```

複数バージョン・通貨ペア・時間軸での重複防止。体系ルール例：

上位2桁 = バージョン (41=v4.1系) / 中位2桁 = 環境番号 / 下位2桁 = 通貨ペアorモード区分

仕様：ユーザーが任意変更可・同一口座内で重複禁止。

EA内部ではMagicNumberをキーにポジションを一括管理。

他MagicNumberのポジションへ干渉せず、決済・トレール・分割決済もMagicNumberで判定。

9. 補足事項・免責

本資料は kutaGOLD 4.x 系EA の開発・運用環境の前提条件を整理した技術文書であり、特定ブローカー・取引環境での再現性を保証するものではありません。

取引条件・サーバー仕様・スワップ値等は随時変更される場合があります。実運用前には必ず最新のブローカー公式情報をご確認ください。

GOLD SMC EA — 総合仕様書 & 引き継ぎ書

Ver.1.1 (SMC統合構造・精密確定版)

作成日：2025年11月

対象：MetaTrader5 / XAUUSD (GOLD)

開発形態：単一EA構成 (外部ファイル依存なし)

本ドキュメントは、GOLD SMC EA
のロジック定義、実装仕様、運用ルール、検証基準を包括した正式版技術仕様兼引き継ぎ書である。

0. 開発目的と設計思想

【目的】

GOLD市場特有の高ボラティリティ・急変動に対応した「構造認識型」自動売買EAを構築する。単なるインジケータ売買ではなく、市場構造（BoS / OB / 再テスト）とキッカーパターン認識に基づく供給・需要ゾーン理論の自動化を目指す。

【設計思想】

- シンプル・堅牢・高再現性
- 複数ファイル構成を避け、EA単体（.mq5）で完結
- バックテストと実運用の再現性を最重要視
- 将来のAI学習適応を見据え、データ構造・ログ構造を明確化

1. システム概要

【基本仕様】

- 通貨ペア：XAUUSD（GOLD）
- 時間足構造：D1（方向確認） / H4（構造判定） / H1（執行）
- 主要構造要素：Break of Structure（BoS） / Order Block（OB） / 再テスト / キッカーパターン
- トレード方向：H4のトレンド方向と一致する方向のみ新規エントリーを許可
- トレード時間帯：ロンドン・NY時間限定（GMT基準で自動補正）
- 実装方式：OnBarベース判定（ティック依存ロジックは使用しない）
- バックテスト方式：Every tick based on real ticks
- ファイル構成：EA単体（MQL5/Experts/kutaGOLD/GOLD_SMC_EA.mq5）

2. 構造ロジック（理論仕様）

2.1 H4構造判定（BoS）

【スイング検出】

- 前後20バーで高値・安値を比較し、相対的なスイング高値 / スイング安値を抽出する。
- 同値の扱い：同値はブレイクなし（静止扱い）とし、BoS判定の対象外とする。

【トレンド定義】

- 2回連続の高値・安値切り上げで上昇トレンド。
- 2回連続の高値・安値切り下げで下降トレンド。

【BoS確定条件】

- 終値がスイング値を $ATR(14) \times 0.1$ 以上明確に突破。
- その状態が2本連続で上抜け / 下抜けとして維持される。

【状態管理】

- 候補BoS 確定BoS 無効BoS（EXPIRED）
- BoS構造体に state（ACTIVE/EXPIRED/TESTED/BROKEN）を持たせて管理する。

【表示（将来の可視化拡張用）】

- 候補：点線
- 確定：実線 + 矢印
- 無効：グレーアウト

2.2 Order Block (OB)

【OB生成条件】

- BoS確定直前3～5本の逆行足を走査する。
- 上昇BoSの場合：最後の下降足
- 下降BoSの場合：最後の上昇足

【OB範囲定義】

- ローソク実体範囲 (Body) のみをOBコア領域とする。
- ヒゲは補助的表示とし、OBの価格帯としては採用しない。
大口約定価格帯 (支配的な実体領域) にフォーカスし、OBが過度に広がることを防ぐ。

【有効期限・テスト管理】

- 新鮮OB：50本
- 古いOB：100本
- テスト済OB：3回テストで削除対象

【テスト判定】

- OB内に価格が侵入し、3本以内に反発した場合：テスト成功。
- 5本以上停滞した場合：弱体化フラグを付与。

【複数OBの重複処理】

- 最新BoSに紐づくOBを優先し、古いBoS由来のOBは EXPIRED 化する。
市場の関心が最新の構造にシフトしている前提で、古いOBを整理しシステム負荷を軽減する。

【表示ルール (将来拡張)】

- 新鮮OB：濃色
- 古いOB：点線
- テスト済：テスト回数に応じて「×1」「×2」等を表示

2.3 キッカーパターン (Kicker)

【定義】

- 弱気ローソクの直後に、ギャップを伴う強気ローソクが出現し、かつ強い実体で前足を包み込むパターン (またはその逆) 。

【検出条件】

- 2本間の終値と始値のギャップが、平均ATRの 0.5 倍を超える。
- 2本目の実体が1本目の実体を完全に包み、かつ下ヒゲ (または上ヒゲ) がほぼ存在しない。

【階層構造とスコアリング】

1. BoS確定時：構造信頼性向上 (+50pt)
2. OB生成時：OB品質評価 (+30pt)
3. OB再侵入時 (再テスト)：トリガー条件 (+20pt)

【スコア加算ルール】

- 各層 (BoS, OB, 再テスト) は独立した根拠として扱い、それぞれの条件を満たした場合にのみ加算する。
- 内包関係にある要素 (例：同一OBに対する再テストの繰り返し) では二重カウントを防止するロジックを実装する。

【スコアとリスクの対応】

- スコア 80以上：リスク 2%（フルポジション）
- スコア 60～79：リスク 1.5%（標準）
- スコア 50～59：リスク 1.0%（軽量）
- スコア 50未満：新規エントリー見送り

2.4 再テスト条件

【価格帯】

- $BoS_Level \pm (ATR \times 0.25)$ 範囲での再接触を再テスト候補とする。

【判定精度】

- 高精度： $\pm ATR \times 0.1$ 即エントリー許可
- 標準： $\pm ATR \times 0.25$ キッカーパターン併用を推奨
- 中精度： $\pm ATR \times 0.5$ 補助条件（時間帯・ボラ状況）必須
- 低精度： $\pm ATR \times 0.75$ 原則無効化推奨

【回数制限】

- 1回目：信頼度100%
- 2回目：信頼度70%（10本以上の間隔がある場合に限定）
- 3回目：信頼度40%（原則エントリー見送り）

2.5 エントリー条件

エントリーは以下の条件を満たした場合のみ許可される。

【必須条件】

- H4トレンド方向とエントリー方向が一致していること。
- H1において、BoSレベルまたはOBコア領域の再テスト後、反発サインが確認されていること。

【補助条件（少なくとも1つ以上）】

- キッカーパターンの出現。
- ATR拡大中（ボラティリティ上昇局面）。
- ロンドン／NYのコア時間帯に一致。

これらを総合スコアとして評価し、所定のスコア閾値（50以上）を満たした場合のみ、新規ポジションを構築する。

2.6 エグジット条件

【Exit条件の優先順位】

1. BoS逆転（構造的条件・最優先）
2. OBブレイク（需給バランス崩壊）
3. ATRベースのリスク管理・時間制限

【構造的条件】

- 逆方向のBoSが確定した場合：全ポジションを即時全決済。
- 既存ポジション方向の主要OBが明確にブレイクされた場合：50%決済。

- 上位時間軸（D1/H4）で逆BoSが発生した場合：75%決済 + トレールを強化。

【リスク条件】

- 初期ストップロス：ATR × 1.5
- トレーリングストップ：ATR × 1.0 を基準としたダイナミックトレール。
- アジア時間跨ぎ / 週末2時間前：ポジション縮小または全決済を強制。

【利益確定条件】

- TP1 = ATR × 1.5（ポジションの30%を利確）
- TP2 = ATR × 3.0（さらに30%を利確）
- TP3 = ATR × 5.0（さらに30%を利確）
- 残り10%はトレールに委ね、トレンドフォロー型のエクステンションを狙う。

3. 実装上の主要対策（7大問題対応）

以下の7つの問題に対して、あらかじめ対策を講じる。

1. 時間軸同期ズレ

階層バー確定 + 同期キュー方式 / 遅延評価で、H4・H1・M15などのバー確定順序を統一。

2. データ肥大化

循環バッファ（リングバッファ） + 状態ベース削除で、古いBoS/OB/ログを自動整理。

3. ATR・出来高異常

多層フォールバック + 異常値除外（スパイク検出）で、異常ボラティリティ時の誤判定を防止。

4. 重複BoS/OB

UUID（識別子） / 親子リンク / フラグ管理により、重複構造の発生と混線を防ぐ。

5. セッションズレ

GMT/DST自動補正 + 境界15分の新規エントリー停止で、ロンドン・NYセッション切替時のブレを抑制。

6. 週末ギャップ誤判定

ATR × 3以上のギャップを検出し、初動バーは待機モードとして構造判定対象外とする。

7. バックテスト差異

Every tick based on real ticks + OnBar統一処理 +
浮動小数点誤差補正で、バックテストとデモの乖離を最小化する。

4. コード骨格（テンプレート構造）

```
//+-----+
//| GOLD_SMC_EA.mq5 (Core Template)
//+-----+
#property strict
#include <Trade/Trade.mqh>
CTrade trade;

//===== Input Settings =====
input double Lots = 0.10;
input int ATR_Period = 14;
```

```

input int Swing_Len_H4 = 20;
input double RiskPerTrade = 1.5;

//===== ENUMS =====//
enum STATE { ACTIVE, EXPIRED, TESTED, BROKEN };

//===== STRUCTS =====//
struct BoS {
    string id;
    datetime time;
    double level;
    bool bullish;
    STATE state;
    double atr;
    int score;
};

struct OB {
    string id;
    datetime time;
    double high, low;
    bool bullish;
    STATE state;
    int testCount;
    string parentBoS;
};

//===== GLOBAL ARRAYS =====//
BoS bos_list[50];
OB ob_list[100];

//===== FUNCTION PROTOTYPES =====//
void CheckBoS();
void CheckOB();
void CheckKicker();
void EvaluateEntry();
void ExitLogic();
double GetATR(string symbol, ENUM_TIMEFRAMES tf, int period);

//===== MAIN ENTRY =====//
int OnInit() { Print("GOLD SMC EA initialized."); return(INIT_SUCCEEDED); }

void OnTick() {
    UpdateTrailingStop();
    MonitorEmergencyExit();
}

int OnCalculate(...) {
    static datetime last_bar=0;
    datetime bar_time=iTime(_Symbol,_Period,0);
    if(bar_time!=last_bar){
        last_bar=bar_time;
        CheckBoS();
        CheckOB();
        CheckKicker();
        EvaluateEntry();
    }
    return(0);
}

```

5. 構造フロー：BoS OB Kicker Entry

【基本フロー】

[BoS検出]

[確定BoS]

OB生成

（同時にキッカー確認）

[プレミアムOB] スコア+30

[再テスト監視]

[キッカー確認層3]

[エントリー実行]

[Exit監視（BoS逆転 / ATR / 時間制限）]

このフローにより、構造認識（BoS） 需給ゾーン（OB） モメンタム（Kicker）
エントリー / エグジットが一貫した論理で接続される。

6. 運用管理ルール

【稼働時間帯】

- 16:00 ~ 02:00 JST（ロンドン ~ NYの高流動時間帯をカバー）。

- 実際の実装では

GMT基準でロンドン / NYセッションの開始・終了を計算し、DST（サマータイム）を自動補正する。

【スプレッド管理】

- スプレッド >

100ポイント（ブローカー仕様に依りて適宜換算）の場合：新規エントリーを自動スキップ。

【重要イベント回避】

- 雇用統計・FOMCなどの主要指標の発表前後30分間は新規エントリーを停止する（実装時は経済指標カレンダー連携または手動設定）。

【週末リスク管理】

- 金曜NY時間 12:00（日本時間 2:00）時点で全ポジションを決済し、週末ギャップリスクを回避する。

7. バックテスト・品質検証

【目標値】

- バックテストとデモの乖離：±5%以内

- 勝率（BoS一致時）：65%以上

- RR比平均：1 : 2.5 以上

- 最大ドローダウン（DD）：15%以下

- トレード頻度：月20～30回（標準）
- ログー致率：95%以上

【基準値の定義】

- 特定期間（推奨：直近12ヶ月）のデモ環境における「月次損益の中央値」を基準値とする。
- 乖離率は以下の式で定義する：

$$\text{乖離率} = |\text{バックテスト損益} - \text{デモ中央値}| \div \text{デモ中央値} \times 100 (\%)$$

- 平均値ではなく中央値（メディアン）を採用することで、外れ値に対する耐性を高め、評価指標の堅牢性を確保する。

8. 補足仕様（Ver.1.1 精密確定版 追記）

以下の7項目は、本EAの再現性・一貫性を高めるために Ver.1.1 で明確化された補足仕様である。

1. BoSのスイング認識ロジック

- 同値はブレイクなし（静止扱い）。
- 厳格なブレイク定義により、レンジ相場でのノイズ発生を防止。

2. Order Block（OB）の範囲定義

- ローソク実体範囲（Body）のみをOBとして採用。
- ヒゲは補助的表示とし、価格帯には含めない。

3. 複数OBの重複処理

- 最新BoSに紐づくOBを優先。
- 古いIOBは EXPIRED 化して無効化。

4. Kickerスコアの多層加算

- BoS、OB、再テストなど各層を独立根拠と見なし、条件を満たした層のみスコア加算。
- 同一OB内での再テストなど、内包関係にある要素は二重カウントを防止。

5. 時間帯とセッション補正

- GMT基準でロンドン・NYの時差を自動補正。
- サマータイム有無にかかわらず、常にコアタイム（高流動帯）を正確に捉える。

6. Exit条件の優先順位

- BoS逆転 > OBブレイク > ATR / 時間制限。
- 構造崩壊（BoS逆転）を最優先に検知し、最大損失回避を重視。

7. 検証ステップ：基準値の定義

- 基準値にはデモ環境12ヶ月間の「月次損益中央値」を採用。
- $\pm 5\%$ 乖離の評価を現実的な安定指標として扱う。

9. 開発ロードマップ（再掲）

【フェーズ別ロードマップ】

1. 基盤構築（1～2週）

- OnBar統一処理の実装。
- GMT / DST補正ロジックの実装。
- テスト環境（デモ口座・バックテスト設定）の整備。

2. データ管理（1週）

- 循環バッファ構造の実装。
- BoS / OB / ログに対するUUID実装。

3. 信頼性強化（1週）

- ATR異常値対策。
- 週末ギャップ除外ロジックの実装。

4. 統合テスト（1～2週）

- 結合テスト・パラメータ最適化。
- バックテストとデモの再現性検証（ $\pm 5\%$ 以内）。

10. 引き継ぎメモ

【ソース情報】

- ファイル名：GOLD_SMC_EA.mq5
- 配置パス：MQL5/Experts/kutaGOLD/

【依存ファイル】

- のみ（外部ライブラリ依存なし）。

【バックテスト設定】

- モデル：Every tick (real ticks)
- データ期間：直近1年
- 使用時間足：H1 または H4

【主要関数呼び出し順序】

- OnCalculate CheckBoS CheckOB CheckKicker EvaluateEntry ExitLogic
- OnTick では、UpdateTrailingStop / MonitorEmergencyExit を実行。

【将来拡張ポイント】

- FVG（Fair Value Gap）検出ロジックの追加（Optional）。
- Sweep判定（Liquidity Grab）の導入。
- AI適応スコアリング（Phase 5予定：PhaseScoreなどとの連携）。

11. 結論

本ドキュメントは、GOLD SMC EA に関する

- ロジック定義（理論 + 実装）
- 実行構造とフロー
- エラー・リスク対策
- テスト・品質評価基準
- 引き継ぎ運用手順

をすべて包括した、正式版技術仕様兼引き継ぎ書である。

本仕様書を前提とすれば、他者がEAを再構築・保守・最適化する場合でも、ロジック・挙動を100%再現することが可能である。

本Ver.1.1は、今後のバージョン展開（Ver.2.x以降）およびAI連携拡張のための安定した基盤仕様として位置づける。

kutaGOLD Titan Series 総合アーカイブ（完全統合版）

1. バージョン進化ライン

v1.0 ~ v3.5 : RSI/ATRシグナル表示 EMA × ADX順張り SMA整合 OOP + AI統合。
最終形 版では EnvAdapt, PhaseScore, AutoMode によりAI自律化。

2. ロジック構造比較

- v1.x : EMA × ADX + BB × RSI (順張り ・ 逆張り統合)
- v1.3 : SMA整合フェーズ理論 (Quarterly Theory)
- v3.x : OOP + 環境適応 + AI (Framework系列)

3. TitanFX実運用仕様

- ECN Zero / USD / レバレッジ500倍
- Spread制御、Cooldown60s、24h稼働
- 推奨足 : M15 (短中期トレンド追従)

4. 核心ロジック (v3.5 時点)

1. EnvAdapt() : D1 ATRで環境評価
2. PhaseScore() : ADX × CHOP × ATRでトレンドAI評価
3. AutoMode() : Safe/Aggressive 自動切替
4. DynamicSL/TP : ATR倍率補正
5. SafetyCheck() : Spread ・ 時間 ・ ポジ重複管理

5. 外部理論との整合

- ORZ手法 : 押し目買い ・ MA3本構造 Quarterly系列に一致
- テクニカル分析 : MA × ADX ・ RSI ・ BB MR/TT系列に由来
- フェーズ理論 : 相場周期識別 PhaseScoreロジックに内包

6. 勝率ロジック分類 (上位10構成)

- SMA押し目型 (Quarterly型)
- MA × RSIコンフルエンス (Mean Revert型)
- BBタッチ順張り + ADX補正 (短期スイング)
- MAクロス + RCI整合 (クロス型強化)
- ATRトレール + MomentumFilter (ティレント継続型)
- ADX付MAクロス (全世代共通)
- PhaseScore = RCI多層判定代替 (AI型)
- Donchian × SMA傾斜判定 (環境適応型)
- EnvAdapt × AutoMode制御 (リスク最適化)
- DynamicCooldown (ボラ制御型再エントリー)

7. 哲学 ・ 設計思想

- シンプル ・ イズ ・ ストロング (Simple is Strong)
- フェーズを読むEA : 状況を数値化し、裁量判断を再現
- AIが人間の「待つ ・ 攻める」を自律的に切り替える

8. 次世代構想 (v5.0 Titan AI Core)

- 強化学習フェーズAI (Phase Learning)
- AutoTune (自己最適化ロット調整)
- PhaseMemory (相場特性の長期記憶)
- DEAL_TIME準拠のタイム制御 (Time*非依存構造)

9. 結論

v1.x = 実験期 v3.x = 構造期 v3.5 = 自律期 v5.x = 自己進化期。
AIと裁量の融合によって、「フェーズを読むEA」として完成。

【修正ログ・再発防止ガイド（原文）】

PyramidCore & kutaGOLD 修正ガイドまとめ

作成者: ktm0429-glitch (AI支援)
作成日: 2025-11-19 11:05:32 (UTC)

目的: 今回発生した PyramidCore.mqh 周りのコンパイルエラーを含む kutaGOLD 3.0PRO の修正ログと、同様の問題が再発したときにすぐこのファイルには (A) kutaGOLD の修正ログと対処法、(B) PyramidCore に対する具体的な修正 (原因・パッチ例・再発防止チェックリス

目次

- 概要 (今回の症状)
- kutaGOLD: 発生した主な問題と対処
- PyramidCore: 具体的エラーと修正内容 (今回のケース)
- 迅速修正フロー (5分で直す手順)
- よく使う置換パターン & コードスニペット
- コミット / PR 用テンプレート
- 再発防止チェックリスト
- 参考 (MetaEditor や MQL5 API の確認ポイント)

概要 (今回の症状)

MetaEditor のコンパイル出力で以下のようなエラーが発生:

- undeclared identifier (例: POSITION_OPEN_PRICE 未定義)
- wrong parameters count / cannot convert enum
- could be one of 2 function(s) (PositionGetDouble のオーバーロード曖昧さ)

原因の要約:

1. 列挙子名の誤用 (MQL5 での正式名称を間違えた)
2. PositionGetDouble の呼び出しが曖昧になりコンパイラがオーバーロードを解決できない
3. クラス内でグローバルAPIと同名、もしくは誤シグネチャのメソッドを定義している (例: iMA)

kutaGOLD — 発生した主な問題 (要点)

1. 同名関数の重複定義
2. プロトタイプのみで実装がない
3. 環境依存API (PositionSelectByIndex など)
4. 未宣言識別子 / パーサー誤認
5. 変数宣言タイミングの問題
6. MQL4/MQL5 API 差異 (StrToDouble vs StringToDouble 等)
7. ソケット・ネットワークのフォールバック不足
8. trade API の戻り値チェック不足
9. イベントハンドラ (OnInit/OnTick/OnDeinit) の重複

対処方針 (要約):

- 重複は削除・統合。ヘッダは宣言のみ。実装は一箇所に。
- Position 列挙は ticket 経由の安全パターンに統一。
- PositionGetDouble 等は出力引数版に統一して曖昧さを回避。
- グローバル API と同名のクラスメソッドは名前変更 (GetMAValue など) して :: を使う。

PyramidCore (今回のケース) — 具体的エラーと修正内容

症状 (MetaEditor 出力の一例)

- undeclared identifier PyramidCore.mqh 271 31
- wrong parameters count PyramidCore.mqh 271 13
- could be one of 2 function(s) PyramidCore.mqh 271 13
 - built-in: double PositionGetDouble(ENUM_POSITION_PROPERTY_DOUBLE)
 - built-in: bool PositionGetDouble(ENUM_POSITION_PROPERTY_DOUBLE,double&)

該当行:

```
```cpp
PositionGetDouble(POSITION_OPEN_PRICE, open_price);
```
```

原因:

- `POSITION_OPEN_PRICE` は MQL5 の列挙子名ではない (正しくは `POSITION_PRICE_OPEN`)。
- `PositionGetDouble` はオーバーロードがあり、呼び出し方によって曖昧になる。戻り値版を使うとコンパイラがどちらのシグネチャを優先するか不明になる。

修正 (確実なパターン) :

- 列挙子名を正しくし、出力引数版で呼び出す。

修正例 :

```
```cpp
double open_price = 0.0;
if(!PositionGetDouble(POSITION_PRICE_OPEN, open_price))
{
 // 取得失敗時のフォールバック処理 (ログ、continue 等)
}
```
```

加えて行った修正 (ファイル全体での整備)

- PositionGetDouble や PositionGetInteger を戻り値版ではなく出力引数版 (成功/失敗を bool で受け取る) へ統一。
- クラス内での iMA 等同名メソッドを `GetMAValue` にリネームし、グローバルの `::iMA` を使ってハンドル取得 CopyBuffer IndicatorRelease
- トレーリングや修正系の関数で PositionGetDouble を用いる際は明示的に変数へ取得し、API の曖昧性を排除。

迅速修正フロー (5分で直す手順)

1. MetaEditor のエラーメッセージで最初エラー行を開く。
2. undeclared identifier 列挙子名/定数名を MetaEditor 補完で確認 (例: POSITION_PRICE_OPEN)。
3. could be one of 2 function(s) 該当関数のプロトタイプを確認。`PositionGetDouble` は出力引数版を使う (PositionGetDouble(prop, double price, ...)
4. wrong parameters count 同名関数の衝突や誤引数。クラス内メソッド名がグローバルAPIと衝突していないか検索し、衝突があればリネーム
5. 修正 再コンパイル 次のエラーへ (この順に潰すのが最速) 。

コマンド (ファイル内検索の例)

- Windows (PowerShell): Select-String -Path *.mqh,*.mq5 -Pattern "PositionGetDouble|POSITION_OPEN_PRICE" -List
- Linux/macOS (ripgrep): rg "PositionGetDouble|POSITION_OPEN_PRICE" -n

よく使う置換パターン (コピペで使える)

- 誤: POSITION_OPEN_PRICE
- 正: POSITION_PRICE_OPEN

- 誤: double profit = PositionGetDouble(POSITION_PROFIT);

正:

```
```cpp
double profit = 0.0;
PositionGetDouble(POSITION_PROFIT, profit);
```
```

- 誤 (クラス内で iMA を直接ラップ) :

```
```cpp
double iMA(...) { /* ... */ }
```
```

正 (リネームして `::iMA` を使用) :

```
```cpp
double GetMAValue(...) {
 int handle = ::iMA(...);
 // CopyBuffer(handle, ...), IndicatorRelease(handle)
}
```
```

代表的な修正スニペット (抜粋)

Position 安全パターン (PositionsTotal ループ) :

```

```mql5
for(int i=0;i<PositionsTotal();i++)
{
 ulong tk = PositionGetTicket(i);
 if(tk==0) continue;
 if(!PositionSelectByTicket(tk)) continue;
 // read position fields
}
```

```

PositionGetDouble の出力引数パターン:

```

```mql5
double p = 0.0;
if(PositionGetDouble(POSITION_PROFIT, p))
{
 // p を使う
}
else
{
 // 取得失敗時の処理
}
```

```

iMA ハンドル取得 CopyBuffer の安全パターン:

```

```mql5
double GetMAValue(string symbol, ENUM_TIMEFRAMES period, int ma_period, int ma_shift,
 ENUM_MA_METHOD method, ENUM_APPLIED_PRICE price, int bar_shift)
{
 int handle = ::iMA(symbol, period, ma_period, ma_shift, method, price);
 if(handle == INVALID_HANDLE) return 0.0;
 double buf[];
 ArraySetAsSeries(buf, true);
 if(CopyBuffer(handle, 0, bar_shift, 1, buf) <= 0) { IndicatorRelease(handle); return 0.0; }
 IndicatorRelease(handle);
 return buf[0];
}
```

```

コミット / PR 用テンプレート (例)

コミットメッセージ例:

Fix(Position): use POSITION_PRICE_OPEN and PositionGetDouble(out) to resolve compile errors

PR 説明例:

- 概要: PyramidCore および関連モジュールで発生していたコンパイルエラーを修正
- 変更点:
 - POSITION_OPEN_PRICE POSITION_PRICE_OPEN に修正
 - PositionGetDouble を出力引数版に統一 (曖昧さを排除)
 - iMA ラッパーを GetMAValue にリネームし ::iMA を使用してバッファ取得
- テスト: MetaEditor コンパイル通過、主要フローの単体確認済み (デモ環境でのフォワード推奨)

再発防止チェックリスト (コミット前に必ず実行)

- [] 列挙子 (POSITION_*, ENUM_*) は MetaEditor の補完で確認したか?
- [] PositionGetDouble/PositionGetInteger 等は出力引数版を使っているか?
- [] グローバル API と同名のクラスメソッドがないか? (iMA, iATR 等)
- [] ハンドル系インジケータのパターン (handle CopyBuffer IndicatorRelease) を守っているか?
- [] すべての trade.* 呼び出しの戻り値をチェックしてログ出力しているか?
- [] ローカル変数は関数先頭で宣言されているか? (互換性のため)
- [] 変更ごとにコンパイルを小まめに行ったか?

参考 (MetaEditor / MQL5 API の確認ポイント)

- MetaEditor のシンボル補完で列挙子名を確認 (POSITION_* 系は類似名が多い)
- MQL5 リファレンス: PositionGetDouble の二つのシグネチャ (戻り値版 / 出力引数版) を常に把握する
- インジケータハンドル API: iMA/iATR のハンドル版と CopyBuffer の使い方を厳守する

Copilot said: エラーが無くなって良かったです。ここまでの経緯を整理して、発生した主なエラー / 警告、原因、どのように対処したか
エラーが無くなって良かったです。ここまでの経緯を整理して、発生した主なエラー / 警告、原因、どのように対処したか (具体的な修

結論 (ひとこと)

根本原因は「名前衝突 (自作ヘルパーが組み込み関数名と同じ)」「誤った API の呼び方 (iATR 等の誤用)」「インクルード/ファイル
発生していた主なエラー / 警告 (一覧)

「file ... not found」

Include ファイルがMetaEditorの検索パス (MQL5\Include) やEAと同じフォルダに無い / 置き場所が間違っている。

「event handling function not found」

.mq5 に必須のイベント関数 (OnInit/OnTick/OnDeinit) がない / コメントアウトされている / ファイルが途中で不完全だった。

「PyramidCore」- unexpected token / declaration without type / many undeclared identifier」系の大量エラー

include が読み込めていない (または読み込めても中身がパースエラー) ため、クラス定義が見つからず連鎖的に出た派生エラー。

「wrong parameters count」/ built-in: int iATR(const string,ENUM_TIMEFRAMES,int)

iATR を「値取得 (シフト付き)」のような形で (4引数で) 呼んでいたため。iATR はハンドルを返す組み込みで、値取得は CopyBuf

「possible loss of data due to type conversion from 'double' to 'int'」

組み込み関数と名前衝突して、意図せず double を返す関数を int に代入していた箇所 (TrendAnalyzer のハンドル取得周りなど)。

「undeclared identifier」の個別ケース (例: POSITION_OPEN_PRICE)

MQL5 の列挙子名が間違っていた (POSITION_OPEN_PRICE は存在せず POSITION_PRICE_OPEN が正しい)。

ヘルパー関数名 (iMA / iBands / iMACD / iATR) と組み込み関数名の衝突

自作ヘルパーが同名で、呼び出しが再帰やパラメータミスマッチを生んでいた。

ファイル先頭の方の非標準構文 (例: input group "..." をそのまま残していた) やエンコーディング (BOM) によるパーサ混乱 MetaEditor の
具体的に行った対処 (何を修正したか)

include ファイルの配置確認・整理

PyramidConfig.mqh / PyramidCore.mqh / RiskManager.mqh / TrendAnalyzer.mqh を MQL5\Include (もしくは EA と同じフォルダ) に正しく
#include を角括弧形式 (#include <...>) に変更してグローバル Include を確実に参照できるようにした (環境に依存する探索問題回避)。

組み込み関数と衝突するヘルパー名をリネーム (衝突回避)

例: iMA / iATR / iBands / iMACD GetMAValue / GetATRValue / GetBandsValue / GetMACDValue

自作ヘルパー内部では組み込み関数を明示 (::iMA, ::iATR など) で呼ぶ / ハンドル取得 CopyBuffer IndicatorRelease の安全パターンに統一
IndicatorRelease を忘れない実装にし、リソースリークを防止。

iATR の誤用を修正 (値のシフト取得は GetATRValue(...) で行い、内部で CopyBuffer(handle, 0, shift, ...) を使うようにした)。

TrendAnalyzer のインジケータハンドル取得を ::iMA / ::iMACD / ::iATR を使って int ハンドルを直接取得するように修正 (double int の警

Position プロパティ取得を明示的に出力引数の PositionGetDouble(..., var) 形式に統一 (曖昧呼出しを排除)。

列挙子名誤記を修正 (例: POSITION_OPEN_PRICE POSITION_PRICE_OPEN)。

メイン EA の非標準記述 (input group ...) を標準 input 宣言 / コメントに置換してパーサエラーを回避。

テスト用 TestInclude.mq5 を用意して include が読み込めるかを確認 (これが通ったことで配置は OK と判断)。

なぜそれで直ったのか (原因と対処の対応表)

include が見つからない コンパイラは型定義を知らない クラス名やポインタ宣言が「unexpected token」になる。対処: ファイルを正

同名ヘルパーの存在 自作ヘルパーが組み込みを上書き/再帰呼び出ししてパラメータ数不一致や型不整合を生む。対処: 名前を変えて

iATR の 4 引数呼び 組み込みシグネチャと違うので「wrong parameters count」。対処: ハンドル取得 + CopyBuffer に置き換え。

double int 警告 ハンドルを返すべき箇所に double を返す誤実装があった。対処: TrendAnalyzer で組み込みハンドル (int) を直接取得

列挙子の名前違い コンパイル時に undeclared identifier。対処: 正しい列挙子 (POSITION_PRICE_OPEN 等) に修正。

残る注意点 (運用上のチェックリスト)

保存エンコーディング: 全ファイルを UTF 8 (BOMなし) で保存する。BOM があるとパーサを壊すケースあり。

include の配置: プロジェクト単位で一貫しておく (EA と同じフォルダ or MQL5\Include)。#include の書き方は統一 ("..."は相対、<...>は

コンパイル前チェックリスト:

OnInit / OnTick / OnDeinit が存在すること。

クラスの定義が .mqh に正しく閉じられている (class ... { ... };) こと。

不要な非コードコメント (`` 等) が混入していないこと。

テスト: まず TestInclude で include を確認、次にコンパイル 警告が残る場合は警告を1つずつ潰す。

パフォーマンス改善の提案 (推奨) 現状の実装では、GetMAValue / GetATRValue / GetBandsValue / GetMACDValue が「呼ばれるたびに:
ハンドルのキャッシュ (おすすめ)

起動時(OnInit)に主要インジケータのハンドルを作成して保持し、OnDeinit で IndicatorRelease する。Get* ヘルパーは既存ハンドルを使っ
例えば TrendAnalyzer のようにハンドルをメンバー変数として持つ方式をメインや PyramidCore にも適用する。

例 (概念スニペット、キャッシュの考え方)

OnInit:

hMA20_15 = iMA(_Symbol, PERIOD_M15, 20, 0, MODE_EMA, PRICE_CLOSE);

GetMAValue: CopyBuffer(hMA20_15, 0, shift, 1, buf)

OnDeinit: IndicatorRelease(hMA20_15)

一括取得の工夫: 必要な複数バッファ (MACD main/signal) を一度に CopyBuffer で取得して使い回す。

kutaGOLD 3.0PRO — 修正ログと対処法まとめ

日時: 2025-11-17

作成者: ktm0429-glitch (開発サポート: AI)

概要

このドキュメントは、今回の修正作業で発生した主なコンパイル/実行エラー、原因、具体的な対処法、及び再発防止チェックリストをまとめたものである。

発生した主な問題（カテゴリ別）

1. 同名関数の重複定義 ("function already defined")
2. プロトタイプのみで実装がない ("must have a body")
3. 環境依存APIの使用 (PositionSelectByIndex 等)
4. 未宣言識別子 / パーサーの誤認 ('10' - some operator expected 等)
5. 変数宣言タイミングの問題 (ローカル変数を関数先頭で宣言していない)
6. MQL4/MQL5 APIの差 (StrToDouble vs StringToDouble)
7. ソケット/ネットワーク失敗時のフォールバック不足
8. trade APIの戻り値チェック不足 (PositionModify/PositionClose)
9. 分割時のイベントハンドラの重複 (OnInit/OnTick/OnDeinit が複数)

各問題の原因と対処（実際に行った修正）

1) 同名関数の重複定義

- 原因: ファイルを統合/分割する過程で同じ関数を複数回コピー・定義してしまった。
- 対処: 重複を削除。関数は1回だけ定義する。分割運用する場合、ヘッダ(.mqh)には宣言のみ、実装は1箇所にする。

2) プロトタイプのみで本体がない (must have a body)

- 原因: 前方宣言を残したまま実装を別ファイル/別位置へ移したためコンパイラが本体を見つけれられない。
- 対処: 実装を同ファイルに追加、またはプロトタイプを削除して実装を1つだけ残す。

3) PositionSelectByIndex 等の非移植API

- 原因: MQL4/MQL5やブローカー実装の差異。PositionSelectByIndex は存在しない環境がある。
- 対処: 安全なパターンに統一：

```
```\nulong ticket = PositionGetTicket(i);\nif(ticket == 0) continue;\nif(!PositionSelectByTicket(ticket)) continue;\n```
```

これが最も移植性が高い。

### ### 4) 未宣言識別子 / パーサー誤認 ("10' - some operator expected")

- 原因: 関数・変数の実装順や欠落したセミコロン、前方宣言の不足など。
- 対処: 実装順を整理し、必要なプロトタイプを用意、構文ミスを修正（括弧/セミコロンの確認）。

### ### 5) 変数宣言タイミング (idx, k 等)

- 原因: 一部の MQL5 コンパイラ設定で関数先頭でのローカル宣言が安全。
- 対処: 各関数の先頭で `int idx=0; int total=0;` などを明示的に宣言・初期化した。

### ### 6) StrToDouble vs StringToDouble

- 原因: MQL4 由来の関数名を残していたためコンパイルエラー。
- 対処: `StrToDouble` を `StringToDouble` に統一 (MQL5 API)。

### ### 7) ソケット (AI) 呼び出しの失敗ハンドリング

- 原因: SocketCreate/SocketConnect/SocketRead が失敗した場合のクリーンアップやフォールバックが不十分。
- 対処: タイムアウトと失敗時に SocketClose を呼び、ダミー応答 (ローカルphase) へフォールバックする。

### ### 8) trade API の戻り値チェック

- 原因: trade.PositionModify/trade.PositionClose が失敗してもログを残していなかった。
- 対処: 失敗時に trade.ResultRetcode() と trade.ResultRetcodeDescription() を PrintFormat で出力するようにした。

### ### 9) 分割時のイベントハンドラ重複

- 原因: 複数ファイルに OnInit/OnDeinit/OnTick を置いてしまった。
- 対処: メイン .mq5 のみがイベントハンドラを持つ。ヘッダ(.mqh) にはイベントハンドラを置かない。

---

## ## 代表的な修正スニペット（抜粋）



- Position 列挙 (安全パターン)

```
```mql5
for(int i=0;i<PositionsTotal();i++)
{
    ulong tk = PositionGetTicket(i);
    if(tk==0) continue;
    if(!PositionSelectByTicket(tk)) continue;
    // read position fields
}
```
```

- ParsePhaseScore: StringToDouble を利用

```
```mql5
string tok = StringSubstr(res, s, e-s);
double val = StringToDouble(tok);
```
```

- Socket: フォールバック

```
```mql5
sock = SocketCreate();
if(sock == INVALID_HANDLE) { g_ai_phase = DummyAI_Phase(g_phase); return; }
if(!SocketConnect(sock, AI_Host, AI_Port, 2000)) { SocketClose(sock); g_ai_phase = DummyAI_Phase(g_phase); return; }
...
if(recv_len <= 0) { SocketClose(sock); g_ai_phase = DummyAI_Phase(g_phase); return; }
```
```

---

## 再発防止チェックリスト (コミット前に必ず実行)

1. 同名関数の重複を全文検索していないか確認する (function名で検索)。
2. OnInit/OnDeinit/OnTick はメインファイルに1つだけ。サブファイルに定義が混入していないか確認。
3. Position 列挙は必ず ticket 経由の安全パターンを使用する。
4. MQL5 API (StringToDouble, CopyBuffer, PositionGetTicket 等) に統一する。
5. すべての trade.\* 呼び出しの戻り値をチェックし、失敗時に結果コードと説明をログする。
6. ループ変数や idx/k/i は関数先頭で明示宣言・初期化する。
7. ソケット通信は必ずタイムアウトとフォールバック (ダミー) を実装する。
8. コンパイルを小まめに実行し、最初に出る数個のエラーを順に潰す。

---