```
//+------------------------------------------------------------------+
//|                                  Universal_Trading_System.mq5 |
//|                                Copyright 2024, Your Company    |
//|                                       https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, Your Company"
#property link      "https://www.mql5.com"
#property version   "1.00"
#property strict

#include <Trade\Trade.mqh>
#include <Trade\PositionInfo.mqh>
#include <Trade\OrderInfo.mqh>

//+------------------------------------------------------------------+
//| Input Parameters                                                 |
//+------------------------------------------------------------------+

//--- Market Selection
input group "=== MARKET SELECTION ==="
input ENUM_MARKET_TYPE MarketType = MARKET_FNO;       // Market Type
input bool EnableForex = true;                        // Enable Forex Trading
input bool EnableCommodities = true;                  // Enable Commodities Trading
input bool EnableIndices = true;                      // Enable Indices Trading

//--- API Configuration
input group "=== API CONFIGURATION ==="
input string FyersAPIKey = "";                        // Fyers API Key
input string FyersSecretKey = "";                     // Fyers Secret Key
input string TrueDataAPIKey = "";                     // TrueData API Key
input string TrueDataUsername = "";                   // TrueData Username
input string TrueDataPassword = "";                   // TrueData Password

//--- Trading Hours
input group "=== TRADING HOURS ==="
input int TradingStartHour = 9;                       // Trading Start Hour
input int TradingStartMinute = 30;                    // Trading Start Minute
input int TradingEndHour = 15;                        // Trading End Hour
input int TradingEndMinute = 29;                      // Trading End Minute
input ENUM_DAY_OF_WEEK TradingStartDay = MONDAY;      // Trading Start Day
input ENUM_DAY_OF_WEEK TradingEndDay = FRIDAY;        // Trading End Day
input bool UseServerTime = false;                     // Use Server Time (false = IST)

//--- Multi-Timeframe Settings
input group "=== TIMEFRAME SETTINGS ==="
input ENUM_TIMEFRAMES MajorTrendTF = PERIOD_D1;       // Major Trend Timeframe
input ENUM_TIMEFRAMES MiddleTrendTF = PERIOD_H1;      // Middle Trend Timeframe
input ENUM_TIMEFRAMES EntryTF = PERIOD_M5;            // Entry Timeframe
```

```
input bool UseDailyForMajor = true;               // Use Daily for Major Trend
input bool Use4HForMajor = false;                 // Use 4H for Major Trend

//--- Indicator Settings
input group "=== ICHIMOKU SETTINGS ==="
input int Tenkan_Period = 9;                      // Tenkan-sen Period
input int Kijun_Period = 26;                      // Kijun-sen Period
input int Senkou_Period = 52;                     // Senkou Span B Period
input int Displacement = 26;                      // Displacement

input group "=== TDI SETTINGS ==="
input int RSI_Period = 13;                        // RSI Period for TDI
input int Volatility_Band = 34;                   // Volatility Band Period
input int RSI_Price_Line = 2;                     // RSI Price Line Period
input int RSI_Signal_Line = 7;                    // RSI Signal Line Period

input group "=== BOLLINGER BANDS ==="
input int BB_Period = 20;                         // Bollinger Bands Period
input double BB_Deviation = 2.0;                  // Bollinger Bands Deviation
input double BB_Squeeze_Threshold = 0.1;          // BB Squeeze Threshold

input group "=== SUPERTREND SETTINGS ==="
input int STR_ENTRY_Period = 20;                  // Supertrend Entry ATR Period
input double STR_ENTRY_Multiplier = 1.0;          // Supertrend Entry ATR Multiplier
input int STR_EXIT_Period = 20;                   // Supertrend Exit ATR Period
input double STR_EXIT_Multiplier = 1.5;           // Supertrend Exit ATR Multiplier

input group "=== SMMA SETTINGS ==="
input int SMMA_Period = 50;                       // SMMA Period

//--- F&O Specific Settings
input group "=== F&O MARKET SETTINGS ==="
input double OI_Bias_Threshold = 30.0;            // OI Bias Threshold (%)
input int MaxSecondaryCharts = 10;                // Max Secondary Charts
input int OTM_Levels_To_Study = 5;                // OTM Levels to Study
input bool EnableHybridLogic = true;              // Enable Hybrid ATM/OTM Logic
input int MaxTradesPerChart = 3;                  // Max Trades per Secondary Chart
input bool AllowBidirectional = false;            // Allow Bidirectional Trading (NRI)

//--- Non-F&O Settings
input group "=== NON-F&O SETTINGS ==="
input int MaxChartsToOpen = 20;                   // Max Charts to Open (Non-F&O)
input bool EnableAutoScanning = true;             // Enable Auto Scanning
input double MinSuccessProbability = 60.0;        // Min Success Probability (%)

//--- Risk Management
input group "=== RISK MANAGEMENT ==="
input double RiskPerTrade = 2.0;                  // Risk Per Trade (%)
```

```
input double MaxDailyLoss = 5.0;                  // Max Daily Loss (%)
input double MaxPositionSize = 0.1;               // Max Position Size (lots)
input int MaxOpenPositions = 10;                  // Max Open Positions
input double StopLossATRMultiplier = 2.0;         // Stop Loss ATR Multiplier
input double TakeProfitRatio = 2.0;               // Take Profit Ratio (Risk:Reward)

//--- Alert Settings
input group "=== ALERT SETTINGS ==="
input bool EnableSoundAlerts = true;              // Enable Sound Alerts
input bool EnableEmailAlerts = true;              // Enable Email Alerts
input bool EnablePhoneAlerts = true;              // Enable Phone Alerts
input string EmailAddress = "pajitmenonai@gmail.com"; // Email Address
input string PhoneNumber = "00971507423656";      // Phone Number
input string SoundFile = "alert.wav";             // Sound File

//--- Display Settings
input group "=== DISPLAY SETTINGS ==="
input bool ShowEconomicCalendar = true;           // Show Economic Calendar
input bool ShowMarketInfo = true;                 // Show Market Info Panel
input bool ShowTradingHours = true;               // Show Trading Hours
input color PanelColor = clrNavy;                 // Panel Color
input color TextColor = clrWhite;                 // Text Color

//+------------------------------------------------------------------+
//| Enumerations                                                     |
//+------------------------------------------------------------------+
enum ENUM_MARKET_TYPE
{
   MARKET_FNO,       // Indian F&O Markets
   MARKET_FOREX,     // Forex Markets
   MARKET_COMMODITIES, // Commodities Markets
   MARKET_INDICES,   // Indices Markets
   MARKET_AUTO       // Auto Detect
};

enum ENUM_TREND_DIRECTION
{
   TREND_UP,
   TREND_DOWN,
   TREND_SIDEWAYS
};

enum ENUM_ENTRY_PATH
{
   PATH_CONTINUATION,
   PATH_PULLBACK,
   PATH_NONE
};
```

```
enum ENUM_CHART_TYPE
{
   CHART_PRIMARY,
   CHART_CALL_SECONDARY,
   CHART_PUT_SECONDARY
};

//+------------------------------------------------------------------+
//| Structure Definitions                                            |
//+------------------------------------------------------------------+
struct MarketInfo
{
   string symbol;
   ENUM_MARKET_TYPE marketType;
   double currentPrice;
   double atmStrike;
   double callOI;
   double putOI;
   double oiBias;
   ENUM_TREND_DIRECTION majorTrend;
   ENUM_TREND_DIRECTION middleTrend;
   ENUM_ENTRY_PATH entryPath;
   datetime lastUpdate;
};

struct TradeSignal
{
   string symbol;
   int signalType; // 1=Buy, -1=Sell
   double entryPrice;
   double stopLoss;
   double takeProfit;
   string comment;
   datetime signalTime;
   bool isValid;
};

struct SecondaryChart
{
   long chartId;
   string symbol;
   ENUM_CHART_TYPE chartType;
   double strikePrice;
   int tradesCount;
   bool isActive;
   datetime createdTime;
};
```

```
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CTrade trade;
CPositionInfo position;
COrderInfo order;

// Indicator Handles
int handleIchimoku;
int handleBB;
int handleSMMA;
int handleSTR_Entry;
int handleSTR_Exit;
int handleATR;

// Arrays for indicator values
double ichimokuTenkan[];
double ichimokuKijun[];
double ichimokuSpanA[];
double ichimokuSpanB[];
double bbUpper[];
double bbMiddle[];
double bbLower[];
double smmaValues[];
double strEntryValues[];
double strExitValues[];
double atrValues[];

// Market data
MarketInfo marketData[];
TradeSignal currentSignals[];
SecondaryChart secondaryCharts[];

// Trading statistics
double dailyPL = 0.0;
int dailyTrades = 0;
datetime lastTradeDate;
double accountBalance;

// Control variables
bool isTradingTime = false;
bool isInitialized = false;
string lastError = "";

//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
```

```
int OnInit()
{
   Print("Initializing Universal Trading System EA...");

   // Initialize trading object
   trade.SetExpertMagicNumber(123456);
   trade.SetMarginMode();
   trade.SetTypeFillingBySymbol(Symbol());

   // Initialize indicators
   if(!InitializeIndicators())
   {
      Print("Failed to initialize indicators");
      return INIT_FAILED;
   }

   // Initialize market data
   if(!InitializeMarketData())
   {
      Print("Failed to initialize market data");
      return INIT_FAILED;
   }

   // Initialize APIs
   if(!InitializeAPIs())
   {
      Print("Failed to initialize APIs");
      return INIT_FAILED;
   }

   // Create display panels
   CreateDisplayPanels();

   // Set arrays as series
   ArraySetAsSeries(ichimokuTenkan, true);
   ArraySetAsSeries(ichimokuKijun, true);
   ArraySetAsSeries(ichimokuSpanA, true);
   ArraySetAsSeries(ichimokuSpanB, true);
   ArraySetAsSeries(bbUpper, true);
   ArraySetAsSeries(bbMiddle, true);
   ArraySetAsSeries(bbLower, true);
   ArraySetAsSeries(smmaValues, true);
   ArraySetAsSeries(strEntryValues, true);
   ArraySetAsSeries(strExitValues, true);
   ArraySetAsSeries(atrValues, true);

   // Get account balance
   accountBalance = AccountInfoDouble(ACCOUNT_BALANCE);
```

```
   isInitialized = true;
   Print("Universal Trading System EA initialized successfully");

   return INIT_SUCCEEDED;
}

//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
{
   Print("Deinitializing Universal Trading System EA. Reason: ", reason);

   // Close secondary charts
   CloseAllSecondaryCharts();

   // Release indicator handles
   ReleaseIndicators();

   // Clean up display objects
   CleanupDisplayObjects();

   Print("Universal Trading System EA deinitialized");
}

//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
{
   if(!isInitialized)
      return;

   // Check trading hours
   isTradingTime = IsTradingTime();
   if(!isTradingTime)
      return;

   // Update daily statistics
   UpdateDailyStats();

   // Check daily loss limit
   if(dailyPL <= -MaxDailyLoss * accountBalance / 100)
   {
      Print("Daily loss limit reached. Stopping trading for today.");
      return;
   }
```

```
   // Main trading logic based on market type
   if(MarketType == MARKET_FNO || MarketType == MARKET_AUTO)
   {
      ProcessFnOMarkets();
   }
   else
   {
      ProcessNonFnOMarkets();
   }

   // Update display panels
   UpdateDisplayPanels();
}

//+------------------------------------------------------------------+
//| Initialize Indicators                                            |
//+------------------------------------------------------------------+
bool InitializeIndicators()
{
   // Ichimoku
   handleIchimoku = iIchimoku(Symbol(), EntryTF, Tenkan_Period, Kijun_Period, Senkou_Period);
   if(handleIchimoku == INVALID_HANDLE)
   {
      Print("Failed to create Ichimoku handle. Error: ", GetLastError());
      return false;
   }

   // Bollinger Bands
   handleBB = iBands(Symbol(), EntryTF, BB_Period, 0, BB_Deviation, PRICE_CLOSE);
   if(handleBB == INVALID_HANDLE)
   {
      Print("Failed to create Bollinger Bands handle. Error: ", GetLastError());
      return false;
   }

   // SMMA
   handleSMMA = iMA(Symbol(), EntryTF, SMMA_Period, 0, MODE_SMMA, PRICE_CLOSE);
   if(handleSMMA == INVALID_HANDLE)
   {
      Print("Failed to create SMMA handle. Error: ", GetLastError());
      return false;
   }

   // ATR for Supertrend calculation
   handleATR = iATR(Symbol(), EntryTF, STR_ENTRY_Period);
   if(handleATR == INVALID_HANDLE)
   {
```

```
      Print("Failed to create ATR handle. Error: ", GetLastError());
      return false;
   }

   return true;
}

//+------------------------------------------------------------------+
//| Initialize Market Data                                           |
//+------------------------------------------------------------------+
bool InitializeMarketData()
{
   ArrayResize(marketData, 0);
   ArrayResize(currentSignals, 0);
   ArrayResize(secondaryCharts, 0);

   return true;
}

//+------------------------------------------------------------------+
//| Initialize APIs                                                  |
//+------------------------------------------------------------------+
bool InitializeAPIs()
{
   if(MarketType == MARKET_FNO)
   {
      // Initialize Fyers API
      if(!InitializeFyersAPI())
      {
         Print("Failed to initialize Fyers API");
         return false;
      }

      // Initialize TrueData API
      if(!InitializeTrueDataAPI())
      {
         Print("Failed to initialize TrueData API");
         return false;
      }
   }

   return true;
}

//+------------------------------------------------------------------+
//| Initialize Fyers API                                             |
//+------------------------------------------------------------------+
bool InitializeFyersAPI()
```

```
{
   if(StringLen(FyersAPIKey) == 0 || StringLen(FyersSecretKey) == 0)
   {
      Print("Fyers API credentials not provided");
      return false;
   }

   // Implementation for Fyers API initialization
   // This would involve HTTP requests to Fyers API endpoints
   Print("Fyers API initialized successfully");
   return true;
}

//+------------------------------------------------------------------+
//| Initialize TrueData API                                          |
//+------------------------------------------------------------------+
bool InitializeTrueDataAPI()
{
   if(StringLen(TrueDataAPIKey) == 0)
   {
      Print("TrueData API credentials not provided");
      return false;
   }

   // Implementation for TrueData API initialization
   Print("TrueData API initialized successfully");
   return true;
}

//+------------------------------------------------------------------+
//| Process F&O Markets                                              |
//+------------------------------------------------------------------+
void ProcessFnOMarkets()
{
   // Scan F&O assets from TrueData feed
   ScanFnOAssets();

   // Process each asset for trend analysis
   for(int i = 0; i < ArraySize(marketData); i++)
   {
      if(marketData[i].marketType != MARKET_FNO)
         continue;

      // Analyze trends
      AnalyzeTrends(marketData[i]);

      // Check for entry conditions
      if(marketData[i].entryPath != PATH_NONE)
```

```
      {
         ProcessEntryConditions(marketData[i]);
      }
   }
}

//+------------------------------------------------------------------+
//| Process Non-F&O Markets                                          |
//+------------------------------------------------------------------+
void ProcessNonFnOMarkets()
{
   // Scan available symbols
   ScanNonFnOAssets();

   // Process current chart
   MarketInfo currentMarket;
   currentMarket.symbol = Symbol();
   currentMarket.marketType = DetermineMarketType(Symbol());
   currentMarket.currentPrice = SymbolInfoDouble(Symbol(), SYMBOL_BID);

   // Analyze trends
   AnalyzeTrends(currentMarket);

   // Check for entry conditions
   if(currentMarket.entryPath != PATH_NONE)
   {
      ProcessEntryConditions(currentMarket);
   }
}

//+------------------------------------------------------------------+
//| Scan F&O Assets                                                  |
//+------------------------------------------------------------------+
void ScanFnOAssets()
{
   // This function would interface with TrueData API to get F&O asset list
   // Implementation would involve HTTP requests to TrueData endpoints

   // For now, adding sample data structure
   if(ArraySize(marketData) == 0)
   {
      ArrayResize(marketData, 5);

      // Sample F&O assets
      marketData[0].symbol = "NIFTY24DEC18500CE";
      marketData[0].marketType = MARKET_FNO;

      marketData[1].symbol = "NIFTY24DEC18500PE";
```

```
      marketData[1].marketType = MARKET_FNO;

      marketData[2].symbol = "BANKNIFTY24DEC45000CE";
      marketData[2].marketType = MARKET_FNO;

      marketData[3].symbol = "BANKNIFTY24DEC45000PE";
      marketData[3].marketType = MARKET_FNO;

      marketData[4].symbol = "RELIANCE24DECFUT";
      marketData[4].marketType = MARKET_FNO;
   }
}

//+------------------------------------------------------------------+
//| Scan Non-F&O Assets                                              |
//+------------------------------------------------------------------+
void ScanNonFnOAssets()
{
   // Implementation for scanning forex, commodities, indices
   // This would rank symbols by success probability
}

//+------------------------------------------------------------------+
//| Analyze Trends                                                   |
//+------------------------------------------------------------------+
void AnalyzeTrends(MarketInfo &market)
{
   // Get major trend from daily/4H timeframe
   market.majorTrend = GetTrendDirection(market.symbol, MajorTrendTF);

   // Get middle trend from 1H/30M timeframe
   market.middleTrend = GetTrendDirection(market.symbol, MiddleTrendTF);

   // Determine entry path
   market.entryPath = DetermineEntryPath(market.majorTrend, market.middleTrend);
}

//+------------------------------------------------------------------+
//| Get Trend Direction                                              |
//+------------------------------------------------------------------+
ENUM_TREND_DIRECTION GetTrendDirection(string symbol, ENUM_TIMEFRAMES timeframe)
{
   // Copy indicator values
   if(CopyBuffer(handleIchimoku, TENKANSEN_LINE, 0, 3, ichimokuTenkan) <= 0)
      return TREND_SIDEWAYS;
   if(CopyBuffer(handleIchimoku, KIJUNSEN_LINE, 0, 3, ichimokuKijun) <= 0)
      return TREND_SIDEWAYS;
   if(CopyBuffer(handleSMMA, 0, 0, 3, smmaValues) <= 0)
```

```
      return TREND_SIDEWAYS;

   double currentPrice = SymbolInfoDouble(symbol, SYMBOL_BID);

   // Determine trend based on multiple conditions
   bool bullish = (ichimokuTenkan[0] > ichimokuKijun[0] &&
                   currentPrice > smmaValues[0] &&
                   ichimokuTenkan[0] > ichimokuTenkan[1]);

   bool bearish = (ichimokuTenkan[0] < ichimokuKijun[0] &&
                   currentPrice < smmaValues[0] &&
                   ichimokuTenkan[0] < ichimokuTenkan[1]);

   if(bullish)
      return TREND_UP;
   else if(bearish)
      return TREND_DOWN;
   else
      return TREND_SIDEWAYS;
}

//+------------------------------------------------------------------+
//| Determine Entry Path                                             |
//+------------------------------------------------------------------+
ENUM_ENTRY_PATH DetermineEntryPath(ENUM_TREND_DIRECTION major, ENUM_TREND_DIRECTION
middle)
{
   // Continuation Path
   if(major == middle && major != TREND_SIDEWAYS)
      return PATH_CONTINUATION;

   // Pullback Path
   if((major == TREND_UP && middle == TREND_DOWN) ||
     (major == TREND_DOWN && middle == TREND_UP))
      return PATH_PULLBACK;

   return PATH_NONE;
}

//+------------------------------------------------------------------+
//| Process Entry Conditions                                         |
//+------------------------------------------------------------------+
void ProcessEntryConditions(MarketInfo &market)
{
   if(market.marketType == MARKET_FNO)
   {
      ProcessFnOEntryConditions(market);
   }
```

```
   else
   {
      ProcessNonFnOEntryConditions(market);
   }
}

//+------------------------------------------------------------------+
//| Process F&O Entry Conditions                                     |
//+------------------------------------------------------------------+
void ProcessFnOEntryConditions(MarketInfo &market)
{
   // Get ATM strike price
   double atmStrike = GetATMStrike(market.symbol, market.currentPrice);
   market.atmStrike = atmStrike;

   // Get Option Chain data
   if(!GetOptionChainData(market))
      return;

   // Apply Hybrid Logic
   if(EnableHybridLogic)
   {
      ApplyHybridLogic(market);
   }
}

//+------------------------------------------------------------------+
//| Get ATM Strike Price                                             |
//+------------------------------------------------------------------+
double GetATMStrike(string symbol, double spotPrice)
{
   // Determine strike interval based on symbol
   double interval = 50; // Default for NIFTY

   if(StringFind(symbol, "BANKNIFTY") >= 0)
      interval = 100;
   else if(StringFind(symbol, "NIFTY") >= 0)
      interval = 50;
   else
      interval = 100; // Stock options

   // Find nearest strike
   double atmStrike = MathRound(spotPrice / interval) * interval;
   return atmStrike;
}

//+------------------------------------------------------------------+
//| Get Option Chain Data                                            |
```

```
//+------------------------------------------------------------------+
bool GetOptionChainData(MarketInfo &market)
{
   // This would interface with APIs to get OI data
   // For now, using sample data
   market.callOI = 100000; // Sample Call OI
   market.putOI = 80000;   // Sample Put OI

   // Calculate OI bias
   double totalOI = market.callOI + market.putOI;
   if(totalOI > 0)
   {
      market.oiBias = ((market.callOI - market.putOI) / totalOI) * 100;
   }

   return true;
}

//+------------------------------------------------------------------+
//| Apply Hybrid Logic                                               |
//+------------------------------------------------------------------+
void ApplyHybridLogic(MarketInfo &market)
{
   double atmStrike = market.atmStrike;
   double callOI = market.callOI;
   double putOI = market.putOI;

   // Path 1: Directional Bias
   bool directionalBiasFound = false;

   // Bullish Directional Pattern
   if(market.oiBias > OI_Bias_Threshold)
   {
      // Check +1 OTM Call bias
      double otmCallOI = GetOTMOptionOI(market.symbol, atmStrike + 50, true); // Call
      double otmCallPutOI = GetOTMOptionOI(market.symbol, atmStrike + 50, false); // Put at same
strike

      if(otmCallOI > otmCallPutOI * (1 + OI_Bias_Threshold/100))
      {
         // Open Call chart at +1 OTM, Put chart at -1 OTM
         OpenSecondaryChart(market.symbol, atmStrike + 50, CHART_CALL_SECONDARY);
         OpenSecondaryChart(market.symbol, atmStrike - 50, CHART_PUT_SECONDARY);
         directionalBiasFound = true;
      }
   }

   // Bearish Directional Pattern
```

```
   if(!directionalBiasFound && market.oiBias < -OI_Bias_Threshold)
   {
      // Check -1 OTM Put bias
      double otmPutOI = GetOTMOptionOI(market.symbol, atmStrike - 50, false); // Put
      double otmPutCallOI = GetOTMOptionOI(market.symbol, atmStrike - 50, true); // Call at same strike

      if(otmPutOI > otmPutCallOI * (1 + OI_Bias_Threshold/100))
      {
         // Open Put chart at -1 OTM, Call chart at +1 OTM
         OpenSecondaryChart(market.symbol, atmStrike - 50, CHART_PUT_SECONDARY);
         OpenSecondaryChart(market.symbol, atmStrike + 50, CHART_CALL_SECONDARY);
         directionalBiasFound = true;
      }
   }

   // Path 2: Independent Bias (if directional not found)
   if(!directionalBiasFound)
   {
      double otmCall1OI = GetOTMOptionOI(market.symbol, atmStrike + 50, true);
      double otmPut1OI = GetOTMOptionOI(market.symbol, atmStrike + 50, false);
      double otmCall2OI = GetOTMOptionOI(market.symbol, atmStrike - 50, true);
      double otmPut2OI = GetOTMOptionOI(market.symbol, atmStrike - 50, false);

      double bias1 = MathAbs(otmCall1OI - otmPut1OI) / (otmCall1OI + otmPut1OI) * 100;
      double bias2 = MathAbs(otmCall2OI - otmPut2OI) / (otmCall2OI + otmPut2OI) * 100;

      if(bias1 > OI_Bias_Threshold && bias2 > OI_Bias_Threshold)
      {
         // Open Call chart at +1 OTM, Put chart at -1 OTM
         OpenSecondaryChart(market.symbol, atmStrike + 50, CHART_CALL_SECONDARY);
         OpenSecondaryChart(market.symbol, atmStrike - 50, CHART_PUT_SECONDARY);
      }
   }
}

//+------------------------------------------------------------------+
//| Get OTM Option OI                                                |
//+------------------------------------------------------------------+
double GetOTMOptionOI(string baseSymbol, double strike, bool isCall)
{
   // This would interface with APIs to get specific option OI
   // For now, returning sample data
   return 50000 + MathRand() % 100000;
}

//+------------------------------------------------------------------+
//| Open Secondary Chart                                             |
//+------------------------------------------------------------------+
```

```
void OpenSecondaryChart(string baseSymbol, double strike, ENUM_CHART_TYPE chartType)
{
   if(ArraySize(secondaryCharts) >= MaxSecondaryCharts)
      return;

   // Construct option symbol
   string optionSymbol = ConstructOptionSymbol(baseSymbol, strike, chartType ==
CHART_CALL_SECONDARY);

   // Create new chart
   long chartId = ChartOpen(optionSymbol, EntryTF);
   if(chartId == 0)
   {
      Print("Failed to open secondary chart for: ", optionSymbol);
      return;
   }

   // Add to secondary charts array
   int index = ArraySize(secondaryCharts);
   ArrayResize(secondaryCharts, index + 1);

   secondaryCharts[index].chartId = chartId;
   secondaryCharts[index].symbol = optionSymbol;
   secondaryCharts[index].chartType = chartType;
   secondaryCharts[index].strikePrice = strike;
   secondaryCharts[index].tradesCount = 0;
   secondaryCharts[index].isActive = true;
   secondaryCharts[index].createdTime = TimeCurrent();

   // Apply EA to secondary chart
   string eaName = MQLInfoString(MQL_PROGRAM_NAME);
   if(!ChartApplyTemplate(chartId, NULL))
   {
      Print("Failed to apply template to secondary chart");
   }

   // Set chart properties for differentiation
   if(chartType == CHART_CALL_SECONDARY)
   {
      ChartSetInteger(chartId, CHART_COLOR_BACKGROUND, clrDarkGreen);
      ObjectCreate(chartId, "CallLabel", OBJ_LABEL, 0, 0, 0);
      ObjectSetString(chartId, "CallLabel", OBJPROP_TEXT, "CALL SIDE");
      ObjectSetInteger(chartId, "CallLabel", OBJPROP_COLOR, clrYellow);
   }
   else
   {
      ChartSetInteger(chartId, CHART_COLOR_BACKGROUND, clrDarkRed);
      ObjectCreate(chartId, "PutLabel", OBJ_LABEL, 0, 0, 0);
```

```
      ObjectSetString(chartId, "PutLabel", OBJPROP_TEXT, "PUT SIDE");
      ObjectSetInteger(chartId, "PutLabel", OBJPROP_COLOR, clrYellow);
   }

   Print("Secondary chart opened: ", optionSymbol, " Type: ",
         chartType == CHART_CALL_SECONDARY ? "CALL" : "PUT");
}

//+------------------------------------------------------------------+
//| Construct Option Symbol                                          |
//+------------------------------------------------------------------+
string ConstructOptionSymbol(string baseSymbol, double strike, bool isCall)
{
   // This function constructs option symbol based on exchange format
   // Sample implementation for NSE format
   string symbol = "";

   if(StringFind(baseSymbol, "NIFTY") >= 0)
   {
      symbol = "NIFTY24DEC" + IntegerToString((int)strike) + (isCall ? "CE" : "PE");
   }
   else if(StringFind(baseSymbol, "BANKNIFTY") >= 0)
   {
      symbol = "BANKNIFTY24DEC" + IntegerToString((int)strike) + (isCall ? "CE" : "PE");
   }

   return symbol;
}

//+------------------------------------------------------------------+
//| Process Non-F&O Entry Conditions                                 |
//+------------------------------------------------------------------+
void ProcessNonFnOEntryConditions(MarketInfo &market)
{
   // Check for buy and sell conditions
   TradeSignal buySignal = CheckBuyConditions(market);
   TradeSignal sellSignal = CheckSellConditions(market);

   if(buySignal.isValid)
   {
      ExecuteTrade(buySignal);
   }

   if(sellSignal.isValid && AllowBidirectional)
   {
      ExecuteTrade(sellSignal);
   }
}
```

```
//+------------------------------------------------------------------+
//| Check Buy Conditions                                             |
//+------------------------------------------------------------------+
TradeSignal CheckBuyConditions(MarketInfo &market)
{
   TradeSignal signal;
   signal.symbol = market.symbol;
   signal.signalType = 1; // Buy
   signal.isValid = false;
   signal.signalTime = TimeCurrent();

   // Get current indicator values
   if(!UpdateIndicatorValues())
      return signal;

   double currentPrice = SymbolInfoDouble(market.symbol, SYMBOL_ASK);

   // 17 Buy Conditions Implementation
   int buyConditionsMet = 0;

   // Condition 1: Ichimoku Bullish
   if(ichimokuTenkan[0] > ichimokuKijun[0] &&
     ichimokuTenkan[1] <= ichimokuKijun[1])
      buyConditionsMet++;

   // Condition 2: Price above Kumo
   if(currentPrice > MathMax(ichimokuSpanA[0], ichimokuSpanB[0]))
      buyConditionsMet++;

   // Condition 3: TK Cross above Kumo
   if(ichimokuTenkan[0] > MathMax(ichimokuSpanA[0], ichimokuSpanB[0]) &&
     ichimokuKijun[0] > MathMax(ichimokuSpanA[0], ichimokuSpanB[0]))
      buyConditionsMet++;

   // Condition 4: Kumo twist bullish
   if(ichimokuSpanA[0] > ichimokuSpanB[0] &&
     ichimokuSpanA[1] <= ichimokuSpanB[1])
      buyConditionsMet++;

   // Condition 5: Price above SMMA
   if(currentPrice > smmaValues[0])
      buyConditionsMet++;

   // Condition 6: SMMA trending up
   if(smmaValues[0] > smmaValues[1])
      buyConditionsMet++;
```

```
   // Condition 7: Bollinger Bands expanding
   double bbWidth = (bbUpper[0] - bbLower[0]) / bbMiddle[0];
   double bbWidthPrev = (bbUpper[1] - bbLower[1]) / bbMiddle[1];
   if(bbWidth > bbWidthPrev)
      buyConditionsMet++;

   // Condition 8: Price above BB Middle
   if(currentPrice > bbMiddle[0])
      buyConditionsMet++;

   // Condition 9: BB Squeeze breakout
   if(bbWidth > BB_Squeeze_Threshold && bbWidthPrev <= BB_Squeeze_Threshold)
      buyConditionsMet++;

   // Condition 10: Supertrend Entry bullish
   double strEntry = CalculateSupertrend(STR_ENTRY_Multiplier, STR_ENTRY_Period, 0);
   if(currentPrice > strEntry)
      buyConditionsMet++;

   // Condition 11-17: Additional technical conditions
   // (Implementation continues with remaining conditions...)

   // Signal is valid if minimum conditions are met
   if(buyConditionsMet >= 8) // Minimum 8 out of 17 conditions
   {
      signal.isValid = true;
      signal.entryPrice = currentPrice;
      signal.stopLoss = currentPrice - (atrValues[0] * StopLossATRMultiplier);
      signal.takeProfit = currentPrice + ((currentPrice - signal.stopLoss) * TakeProfitRatio);
      signal.comment = "Buy Signal - " + IntegerToString(buyConditionsMet) + " conditions met";
   }

   return signal;
}

//+------------------------------------------------------------------+
//| Check Sell Conditions                                            |
//+------------------------------------------------------------------+
TradeSignal CheckSellConditions(MarketInfo &market)
{
   TradeSignal signal;
   signal.symbol = market.symbol;
   signal.signalType = -1; // Sell
   signal.isValid = false;
   signal.signalTime = TimeCurrent();

   // Get current indicator values
   if(!UpdateIndicatorValues())
```

```
      return signal;

   double currentPrice = SymbolInfoDouble(market.symbol, SYMBOL_BID);

   // 17 Sell Conditions Implementation (opposite of buy conditions)
   int sellConditionsMet = 0;

   // Condition 1: Ichimoku Bearish
   if(ichimokuTenkan[0] < ichimokuKijun[0] &&
     ichimokuTenkan[1] >= ichimokuKijun[1])
      sellConditionsMet++;

   // Condition 2: Price below Kumo
   if(currentPrice < MathMin(ichimokuSpanA[0], ichimokuSpanB[0]))
      sellConditionsMet++;

   // Additional sell conditions implementation...
   // (Similar structure to buy conditions but for bearish signals)

   // Signal is valid if minimum conditions are met
   if(sellConditionsMet >= 8) // Minimum 8 out of 17 conditions
   {
      signal.isValid = true;
      signal.entryPrice = currentPrice;
      signal.stopLoss = currentPrice + (atrValues[0] * StopLossATRMultiplier);
      signal.takeProfit = currentPrice - ((signal.stopLoss - currentPrice) * TakeProfitRatio);
      signal.comment = "Sell Signal - " + IntegerToString(sellConditionsMet) + " conditions met";
   }

   return signal;
}

//+------------------------------------------------------------------+
//| Update Indicator Values                                          |
//+------------------------------------------------------------------+
bool UpdateIndicatorValues()
{
   // Copy all indicator buffers
   if(CopyBuffer(handleIchimoku, TENKANSEN_LINE, 0, 3, ichimokuTenkan) <= 0)
      return false;
   if(CopyBuffer(handleIchimoku, KIJUNSEN_LINE, 0, 3, ichimokuKijun) <= 0)
      return false;
   if(CopyBuffer(handleIchimoku, SENKOUSPANA_LINE, 0, 3, ichimokuSpanA) <= 0)
      return false;
   if(CopyBuffer(handleIchimoku, SENKOUSPANB_LINE, 0, 3, ichimokuSpanB) <= 0)
      return false;
   if(CopyBuffer(handleBB, UPPER_BAND, 0, 3, bbUpper) <= 0)
      return false;
```

```
   if(CopyBuffer(handleBB, BASE_LINE, 0, 3, bbMiddle) <= 0)
      return false;
   if(CopyBuffer(handleBB, LOWER_BAND, 0, 3, bbLower) <= 0)
      return false;
   if(CopyBuffer(handleSMMA, 0, 0, 3, smmaValues) <= 0)
      return false;
   if(CopyBuffer(handleATR, 0, 0, 3, atrValues) <= 0)
      return false;

   return true;
}

//+------------------------------------------------------------------+
//| Calculate Supertrend                                             |
//+------------------------------------------------------------------+
double CalculateSupertrend(double multiplier, int period, int shift)
{
   double hl2 = (iHigh(Symbol(), EntryTF, shift) + iLow(Symbol(), EntryTF, shift)) / 2;
   double atr = atrValues[shift];
   double upperBand = hl2 + (multiplier * atr);
   double lowerBand = hl2 - (multiplier * atr);

   // Simplified Supertrend calculation
   double close = iClose(Symbol(), EntryTF, shift);
   if(close > upperBand)
      return lowerBand;
   else
      return upperBand;
}

//+------------------------------------------------------------------+
//| Execute Trade                                                    |
//+------------------------------------------------------------------+
void ExecuteTrade(TradeSignal &signal)
{
   // Check position limits
   if(PositionsTotal() >= MaxOpenPositions)
   {
      Print("Maximum positions limit reached");
      return;
   }

   // Calculate position size
   double lotSize = CalculatePositionSize(signal);
   if(lotSize <= 0)
      return;

   // Execute trade
```

```
   bool result = false;
   if(signal.signalType > 0) // Buy
   {
      result = trade.Buy(lotSize, signal.symbol, signal.entryPrice,
                         signal.stopLoss, signal.takeProfit, signal.comment);
   }
   else // Sell
   {
      result = trade.Sell(lotSize, signal.symbol, signal.entryPrice,
                          signal.stopLoss, signal.takeProfit, signal.comment);
   }

   if(result)
   {
      dailyTrades++;
      SendAlerts("Trade Executed", signal.comment + " on " + signal.symbol);
      Print("Trade executed: ", signal.comment);
   }
   else
   {
      int error = GetLastError();
      Print("Trade execution failed. Error: ", error, " - ", ErrorDescription(error));
      lastError = "Trade execution error: " + IntegerToString(error);
   }
}

//+------------------------------------------------------------------+
//| Calculate Position Size                                          |
//+------------------------------------------------------------------+
double CalculatePositionSize(TradeSignal &signal)
{
   double balance = AccountInfoDouble(ACCOUNT_BALANCE);
   double riskAmount = balance * RiskPerTrade / 100;

   double stopDistance = MathAbs(signal.entryPrice - signal.stopLoss);
   if(stopDistance <= 0)
      return 0;

   double tickValue = SymbolInfoDouble(signal.symbol, SYMBOL_TRADE_TICK_VALUE);
   double tickSize = SymbolInfoDouble(signal.symbol, SYMBOL_TRADE_TICK_SIZE);
   double lotStep = SymbolInfoDouble(signal.symbol, SYMBOL_VOLUME_STEP);

   double lotSize = (riskAmount / (stopDistance / tickSize * tickValue));
   lotSize = MathFloor(lotSize / lotStep) * lotStep;

   // Apply maximum position size limit
   if(lotSize > MaxPositionSize)
      lotSize = MaxPositionSize;
```

```
   return lotSize;
}

//+------------------------------------------------------------------+
//| Send Alerts                                                      |
//+------------------------------------------------------------------+
void SendAlerts(string title, string message)
{
   string alertMessage = title + ": " + message + " at " + TimeToString(TimeCurrent());

   if(EnableSoundAlerts)
   {
      PlaySound(SoundFile);
   }

   if(EnableEmailAlerts)
   {
      SendMail(title, alertMessage);
   }

   if(EnablePhoneAlerts)
   {
      SendNotification(alertMessage);
   }

   Print("Alert sent: ", alertMessage);
}

//+------------------------------------------------------------------+
//| Check Trading Time                                               |
//+------------------------------------------------------------------+
bool IsTradingTime()
{
   MqlDateTime dt;
   TimeCurrent(dt);

   // Check day of week
   if(dt.day_of_week < TradingStartDay || dt.day_of_week > TradingEndDay)
      return false;

   // Check time
   int currentMinutes = dt.hour * 60 + dt.min;
   int startMinutes = TradingStartHour * 60 + TradingStartMinute;
   int endMinutes = TradingEndHour * 60 + TradingEndMinute;

   return (currentMinutes >= startMinutes && currentMinutes <= endMinutes);
}
```

```
//+------------------------------------------------------------------+
//| Update Daily Statistics                                          |
//+------------------------------------------------------------------+
void UpdateDailyStats()
{
   MqlDateTime dt;
   TimeCurrent(dt);

   // Reset daily stats at start of new day
   if(TimeToStruct(lastTradeDate, dt) && dt.day != dt.day)
   {
      dailyPL = 0.0;
      dailyTrades = 0;
   }

   lastTradeDate = TimeCurrent();

   // Calculate current daily P&L
   dailyPL = 0;
   for(int i = 0; i < PositionsTotal(); i++)
   {
      if(position.SelectByIndex(i))
      {
         if(TimeToStruct(position.Time(), dt) && dt.day == dt.day)
         {
            dailyPL += position.Profit();
         }
      }
   }
}

//+------------------------------------------------------------------+
//| Determine Market Type                                            |
//+------------------------------------------------------------------+
ENUM_MARKET_TYPE DetermineMarketType(string symbol)
{
   if(StringFind(symbol, "CE") > 0 || StringFind(symbol, "PE") > 0 || StringFind(symbol, "FUT") > 0)
      return MARKET_FNO;
   else if(StringFind(symbol, "USD") >= 0 || StringFind(symbol, "EUR") >= 0 ||
           StringFind(symbol, "GBP") >= 0 || StringFind(symbol, "JPY") >= 0)
      return MARKET_FOREX;
   else
      return MARKET_COMMODITIES;
}

//+------------------------------------------------------------------+
//| Create Display Panels                                            |
```

```
//+------------------------------------------------------------------+
void CreateDisplayPanels()
{
   if(!ShowMarketInfo)
      return;

   // Create main info panel
   ObjectCreate(0, "InfoPanel", OBJ_RECTANGLE_LABEL, 0, 0, 0);
   ObjectSetInteger(0, "InfoPanel", OBJPROP_XDISTANCE, 10);
   ObjectSetInteger(0, "InfoPanel", OBJPROP_YDISTANCE, 10);
   ObjectSetInteger(0, "InfoPanel", OBJPROP_XSIZE, 300);
   ObjectSetInteger(0, "InfoPanel", OBJPROP_YSIZE, 200);
   ObjectSetInteger(0, "InfoPanel", OBJPROP_COLOR, PanelColor);
   ObjectSetInteger(0, "InfoPanel", OBJPROP_BGCOLOR, PanelColor);
   ObjectSetInteger(0, "InfoPanel", OBJPROP_BORDER_COLOR, clrWhite);

   // Create text labels
   ObjectCreate(0, "InfoText", OBJ_LABEL, 0, 0, 0);
   ObjectSetInteger(0, "InfoText", OBJPROP_XDISTANCE, 15);
   ObjectSetInteger(0, "InfoText", OBJPROP_YDISTANCE, 15);
   ObjectSetInteger(0, "InfoText", OBJPROP_COLOR, TextColor);
   ObjectSetString(0, "InfoText", OBJPROP_TEXT, "Universal Trading System");
   ObjectSetString(0, "InfoText", OBJPROP_FONT, "Arial");
   ObjectSetInteger(0, "InfoText", OBJPROP_FONTSIZE, 12);
}

//+------------------------------------------------------------------+
//| Update Display Panels                                            |
//+------------------------------------------------------------------+
void UpdateDisplayPanels()
{
   if(!ShowMarketInfo)
      return;

   string infoText = "Universal Trading System\n";
   infoText += "Market Type: " + EnumToString(MarketType) + "\n";
   infoText += "Trading Time: " + (isTradingTime ? "ACTIVE" : "INACTIVE") + "\n";
   infoText += "Daily P&L: " + DoubleToString(dailyPL, 2) + "\n";
   infoText += "Daily Trades: " + IntegerToString(dailyTrades) + "\n";
   infoText += "Open Positions: " + IntegerToString(PositionsTotal()) + "\n";
   infoText += "Secondary Charts: " + IntegerToString(ArraySize(secondaryCharts)) + "\n";

   if(StringLen(lastError) > 0)
      infoText += "Last Error: " + lastError + "\n";

   ObjectSetString(0, "InfoText", OBJPROP_TEXT, infoText);
}
```

```
//+------------------------------------------------------------------+
//| Close All Secondary Charts                                       |
//+------------------------------------------------------------------+
void CloseAllSecondaryCharts()
{
   for(int i = 0; i < ArraySize(secondaryCharts); i++)
   {
      if(secondaryCharts[i].isActive)
      {
         ChartClose(secondaryCharts[i].chartId);
      }
   }
   ArrayResize(secondaryCharts, 0);
}

//+------------------------------------------------------------------+
//| Release Indicators                                               |
//+------------------------------------------------------------------+
void ReleaseIndicators()
{
   if(handleIchimoku != INVALID_HANDLE)
      IndicatorRelease(handleIchimoku);
   if(handleBB != INVALID_HANDLE)
      IndicatorRelease(handleBB);
   if(handleSMMA != INVALID_HANDLE)
      IndicatorRelease(handleSMMA);
   if(handleATR != INVALID_HANDLE)
      IndicatorRelease(handleATR);
}

//+------------------------------------------------------------------+
//| Cleanup Display Objects                                          |
//+------------------------------------------------------------------+
void CleanupDisplayObjects()
{
   ObjectDelete(0, "InfoPanel");
   ObjectDelete(0, "InfoText");
   // Delete other display objects
}

//+------------------------------------------------------------------+
//| Error Description Function                                       |
//+------------------------------------------------------------------+
string ErrorDescription(int error_code)
{
   string error_string;

   switch(error_code)
```

```
{
   case 4000: error_string="No error returned"; break;
   case 4001: error_string="Wrong function pointer"; break;
   case 4002: error_string="Array index is out of range"; break;
   case 4003: error_string="No memory for function call stack"; break;
   case 4004: error_string="Recursive stack overflow"; break;
   case 4005: error_string="Not enough stack for parameter"; break;
   case 4006: error_string="No memory for parameter string"; break;
   case 4007: error_string="No memory for temp string"; break;
   case 4008: error_string="Non-initialized string"; break;
   case 4009: error_string="Non-initialized string in array"; break;
   case 4010: error_string="No memory for array string"; break;
   case 4011: error_string="Too long string"; break;
   case 4012: error_string="Remainder from zero divide"; break;
   case 4013: error_string="Zero divide"; break;
   case 4014: error_string="Unknown command"; break;
   case 4015: error_string="Wrong jump"; break;
   case 4016: error_string="Non-initialized array"; break;
   case 4017: error_string="DLL calls are not allowed"; break;
   case 4018: error_string="Cannot load library"; break;
   case 4019: error_string="Cannot call function"; break;
   case 4020: error_string="Expert function calls are not allowed"; break;
   case 4021: error_string="Not enough memory for temp string returned from function"; break;
   case 4022: error_string="System is busy"; break;
   case 4050: error_string="Invalid function parameters count"; break;
   case 4051: error_string="Invalid function parameter value"; break;
   case 4052: error_string="String function internal error"; break;
   case 4053: error_string="Some array error"; break;
   case 4054: error_string="Incorrect series array using"; break;
   case 4055: error_string="Custom indicator error"; break;
   case 4056: error_string="Arrays are incompatible"; break;
   case 4057: error_string="Global variables processing error"; break;
   case 4058: error_string="Global variable not found"; break;
   case 4059: error_string="Function is not allowed in testing mode"; break;
   case 4060: error_string="Function is not confirmed"; break;
   case 4061: error_string="Send mail error"; break;
   case 4062: error_string="String parameter expected"; break;
   case 4063: error_string="Integer parameter expected"; break;
   case 4064: error_string="Double parameter expected"; break;
   case 4065: error_string="Array as parameter expected"; break;
   case 4066: error_string="Requested history data in update state"; break;
   case 4067: error_string="Internal trade error"; break;
   case 4068: error_string="Resource not found"; break;
   case 4069: error_string="Resource not supported"; break;
   case 4070: error_string="Duplicate resource"; break;
   case 4071: error_string="Custom indicator cannot initialize"; break;
   case 4072: error_string="Cannot load custom indicator"; break;
   case 4073: error_string="No history data"; break;
```

```
      case 4074: error_string="No memory for history data"; break;
      case 4075: error_string="Not enough memory for indicator calculation"; break;
      default:   error_string="Unknown error";
   }

   return error_string;
}

//+------------------------------------------------------------------+
//| Timer Function                                                   |
//+------------------------------------------------------------------+
void OnTimer()
{
   // Update market data every minute
   if(MarketType == MARKET_FNO)
   {
      UpdateFnOMarketData();
   }

   // Check for exit conditions
   CheckExitConditions();

   // Update display panels
   UpdateDisplayPanels();
}

//+------------------------------------------------------------------+
//| Update F&O Market Data                                           |
//+------------------------------------------------------------------+
void UpdateFnOMarketData()
{
   // This function would update market data from APIs
   // Implementation would involve periodic API calls
}

//+------------------------------------------------------------------+
//| Check Exit Conditions                                            |
//+------------------------------------------------------------------+
void CheckExitConditions()
{
   for(int i = 0; i < PositionsTotal(); i++)
   {
      if(position.SelectByIndex(i))
      {
         // Check STR-EXIT based exit
         double strExit = CalculateSupertrend(STR_EXIT_Multiplier, STR_EXIT_Period, 0);
         double currentPrice = position.PriceCurrent();
```

```
         bool shouldExit = false;

         if(position.PositionType() == POSITION_TYPE_BUY)
         {
            if(currentPrice < strExit)
               shouldExit = true;
         }
         else if(position.PositionType() == POSITION_TYPE_SELL)
         {
            if(currentPrice > strExit)
               shouldExit = true;
         }

         if(shouldExit)
         {
            if(trade.PositionClose(position.Ticket()))
            {
               SendAlerts("Position Closed", "STR-EXIT signal for " + position.Symbol());
            }
         }
      }
   }
}

//+------------------------------------------------------------------+
```