//+------------------------------------------------------------------+

//| HLHB TRADING SYSTEM.mq5 |

//| Copyright 2020, MetaQuotes Software Corp. |

//| https://www.mql5.com |

//+------------------------------------------------------------------+

#property copyright "Copyright 2020, MetaQuotes Software Corp."

#property link "https://www.mql5.com"

#property version "1.00"

//+------------------------------------------------------------------+

//| Include |

//+------------------------------------------------------------------+

#include <Expert\Expert.mqh>

//--- available signals

#include <Expert\Signal\SignalMA.mqh>

#include <Expert\Signal\SignalStoch.mqh>

//--- available trailing

#include <Expert\Trailing\TrailingNone.mqh>

//--- available money management

#include <Expert\Money\MoneyFixedLot.mqh>

//+------------------------------------------------------------------+

//| Inputs |

//+------------------------------------------------------------------+

//--- inputs for expert

input string Expert\_Title ="HLHB TRADING SYSTEM"; // Document name

ulong Expert\_MagicNumber =32396; //

bool Expert\_EveryTick =false; //

//--- inputs for main signal

input int Signal\_ThresholdOpen =10; // Signal threshold value to open [0...100]

input int Signal\_ThresholdClose =10; // Signal threshold value to close [0...100]

input double Signal\_PriceLevel =0.0; // Price level to execute a deal

input double Signal\_StopLevel =50.0; // Stop Loss level (in points)

input double Signal\_TakeLevel =50.0; // Take Profit level (in points)

input int Signal\_Expiration =4; // Expiration of pending orders (in bars)

input int Signal\_0\_MA\_PeriodMA =10; // Moving Average(10,0,...) Period of averaging

input int Signal\_0\_MA\_Shift =0; // Moving Average(10,0,...) Time shift

input ENUM\_MA\_METHOD Signal\_0\_MA\_Method =MODE\_EMA; // Moving Average(10,0,...) Method of averaging

input ENUM\_APPLIED\_PRICE Signal\_0\_MA\_Applied =PRICE\_CLOSE; // Moving Average(10,0,...) Prices series

input double Signal\_0\_MA\_Weight =1.0; // Moving Average(10,0,...) Weight [0...1.0]

input int Signal\_1\_MA\_PeriodMA =20; // Moving Average(20,0,...) Period of averaging

input int Signal\_1\_MA\_Shift =0; // Moving Average(20,0,...) Time shift

input ENUM\_MA\_METHOD Signal\_1\_MA\_Method =MODE\_EMA; // Moving Average(20,0,...) Method of averaging

input ENUM\_APPLIED\_PRICE Signal\_1\_MA\_Applied =PRICE\_CLOSE; // Moving Average(20,0,...) Prices series

input double Signal\_1\_MA\_Weight =1.0; // Moving Average(20,0,...) Weight [0...1.0]

input int Signal\_Stoch\_PeriodK =14; // Stochastic(14,3,3,...) K-period

input int Signal\_Stoch\_PeriodD =3; // Stochastic(14,3,3,...) D-period

input int Signal\_Stoch\_PeriodSlow=3; // Stochastic(14,3,3,...) Period of slowing

input ENUM\_STO\_PRICE Signal\_Stoch\_Applied =STO\_LOWHIGH; // Stochastic(14,3,3,...) Prices to apply to

input double Signal\_Stoch\_Weight =1.0; // Stochastic(14,3,3,...) Weight [0...1.0]

//--- inputs for money

input double Money\_FixLot\_Percent =10.0; // Percent

input double Money\_FixLot\_Lots =0.1; // Fixed volume

//+------------------------------------------------------------------+

//| Global expert object |

//+------------------------------------------------------------------+

CExpert ExtExpert;

//+------------------------------------------------------------------+

//| Initialization function of the expert |

//+------------------------------------------------------------------+

int OnInit()

{

//--- Initializing expert

if(!ExtExpert.Init(Symbol(),Period(),Expert\_EveryTick,Expert\_MagicNumber))

{

//--- failed

printf(\_\_FUNCTION\_\_+": error initializing expert");

ExtExpert.Deinit();

return(INIT\_FAILED);

}

//--- Creating signal

CExpertSignal \*signal=new CExpertSignal;

if(signal==NULL)

{

//--- failed

printf(\_\_FUNCTION\_\_+": error creating signal");

ExtExpert.Deinit();

return(INIT\_FAILED);

}

//---

ExtExpert.InitSignal(signal);

signal.ThresholdOpen(Signal\_ThresholdOpen);

signal.ThresholdClose(Signal\_ThresholdClose);

signal.PriceLevel(Signal\_PriceLevel);

signal.StopLevel(Signal\_StopLevel);

signal.TakeLevel(Signal\_TakeLevel);

signal.Expiration(Signal\_Expiration);

//--- Creating filter CSignalMA

CSignalMA \*filter0=new CSignalMA;

if(filter0==NULL)

{

//--- failed

printf(\_\_FUNCTION\_\_+": error creating filter0");

ExtExpert.Deinit();

return(INIT\_FAILED);

}

signal.AddFilter(filter0);

//--- Set filter parameters

filter0.PeriodMA(Signal\_0\_MA\_PeriodMA);

filter0.Shift(Signal\_0\_MA\_Shift);

filter0.Method(Signal\_0\_MA\_Method);

filter0.Applied(Signal\_0\_MA\_Applied);

filter0.Weight(Signal\_0\_MA\_Weight);

//--- Creating filter CSignalMA

CSignalMA \*filter1=new CSignalMA;

if(filter1==NULL)

{

//--- failed

printf(\_\_FUNCTION\_\_+": error creating filter1");

ExtExpert.Deinit();

return(INIT\_FAILED);

}

signal.AddFilter(filter1);

//--- Set filter parameters

filter1.PeriodMA(Signal\_1\_MA\_PeriodMA);

filter1.Shift(Signal\_1\_MA\_Shift);

filter1.Method(Signal\_1\_MA\_Method);

filter1.Applied(Signal\_1\_MA\_Applied);

filter1.Weight(Signal\_1\_MA\_Weight);

//--- Creating filter CSignalStoch

CSignalStoch \*filter2=new CSignalStoch;

if(filter2==NULL)

{

//--- failed

printf(\_\_FUNCTION\_\_+": error creating filter2");

ExtExpert.Deinit();

return(INIT\_FAILED);

}

signal.AddFilter(filter2);

//--- Set filter parameters

filter2.PeriodK(Signal\_Stoch\_PeriodK);

filter2.PeriodD(Signal\_Stoch\_PeriodD);

filter2.PeriodSlow(Signal\_Stoch\_PeriodSlow);

filter2.Applied(Signal\_Stoch\_Applied);

filter2.Weight(Signal\_Stoch\_Weight);

//--- Creation of trailing object

CTrailingNone \*trailing=new CTrailingNone;

if(trailing==NULL)

{

//--- failed

printf(\_\_FUNCTION\_\_+": error creating trailing");

ExtExpert.Deinit();

return(INIT\_FAILED);

}

//--- Add trailing to expert (will be deleted automatically))

if(!ExtExpert.InitTrailing(trailing))

{

//--- failed

printf(\_\_FUNCTION\_\_+": error initializing trailing");

ExtExpert.Deinit();

return(INIT\_FAILED);

}

//--- Set trailing parameters

//--- Creation of money object

CMoneyFixedLot \*money=new CMoneyFixedLot;

if(money==NULL)

{

//--- failed

printf(\_\_FUNCTION\_\_+": error creating money");

ExtExpert.Deinit();

return(INIT\_FAILED);

}

//--- Add money to expert (will be deleted automatically))

if(!ExtExpert.InitMoney(money))

{

//--- failed

printf(\_\_FUNCTION\_\_+": error initializing money");

ExtExpert.Deinit();

return(INIT\_FAILED);

}

//--- Set money parameters

money.Percent(Money\_FixLot\_Percent);

money.Lots(Money\_FixLot\_Lots);

//--- Check all trading objects parameters

if(!ExtExpert.ValidationSettings())

{

//--- failed

ExtExpert.Deinit();

return(INIT\_FAILED);

}

//--- Tuning of all necessary indicators

if(!ExtExpert.InitIndicators())

{

//--- failed

printf(\_\_FUNCTION\_\_+": error initializing indicators");

ExtExpert.Deinit();

return(INIT\_FAILED);

}

//--- ok

return(INIT\_SUCCEEDED);

}

//+------------------------------------------------------------------+

//| Deinitialization function of the expert |

//+------------------------------------------------------------------+

void OnDeinit(const int reason)

{

ExtExpert.Deinit();

}

//+------------------------------------------------------------------+

//| "Tick" event handler function |

//+------------------------------------------------------------------+

void OnTick()

{

ExtExpert.OnTick();

}

//+------------------------------------------------------------------+

//| "Trade" event handler function |

//+------------------------------------------------------------------+

void OnTrade()

{

ExtExpert.OnTrade();

}

//+------------------------------------------------------------------+

//| "Timer" event handler function |

//+------------------------------------------------------------------+

void OnTimer()

{

ExtExpert.OnTimer();

}

//+------------------------------------------------------------------+