

```
#include "sierrachart.h"
```

```
/*=====*/
SCSFExport scsf_TradingSystemBasedOnAlertCondition(SCStudyInterfaceRef sc)
{
    SCInputRef Input_Enabled = sc.Input[0];
    SCInputRef Input_NumBarsToCalculate = sc.Input[1];

    SCInputRef Input_DrawStyleOffsetType = sc.Input[2];
    SCInputRef Input_PercentageOrTicksOffset = sc.Input[3];

    SCInputRef Input_OrderActionOnAlert = sc.Input[4];

    SCInputRef Input_EvaluateOnBarCloseOnly = sc.Input[5];
    SCInputRef Input_SendTradeOrdersToTradeService = sc.Input[6];
    SCInputRef Input_MaximumPositionAllowed = sc.Input[7];
    SCInputRef Input_MaximumLongTradesPerDay = sc.Input[8];
    SCInputRef Input_MaximumShortTradesPerDay = sc.Input[9];
    SCInputRef Input_EnableDebuggingOutput = sc.Input[10];
    SCInputRef Input_CancelAllWorkingOrdersOnExit = sc.Input[11];

    SCInputRef Input_AllowTradingOnlyDuringTimeRange = sc.Input[12];
    SCInputRef Input_StartTimeForAllowedTimeRange = sc.Input[13];
    SCInputRef Input_EndTimeForAllowedTimeRange = sc.Input[14];
    SCInputRef Input_AllowOnlyOneTradePerBar = sc.Input[15];
    SCInputRef Input_Version = sc.Input[16];
    SCInputRef Input_SupportReversals = sc.Input[17];
    SCInputRef Input_AllowMultipleEntriesInSameDirection = sc.Input[18];
    SCInputRef Input_AllowEntryWithWorkingOrders = sc.Input[19];
    SCInputRef Input_ControlBarButtonNumberForEnableDisable = sc.Input[20];
    SCInputRef Input_CancelAllOrdersOnEntries = sc.Input[21];
    SCInputRef Input_CancelAllOrdersOnReversals = sc.Input[22];

    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[1];

    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "Trading System Based on Alert Condition";

        sc.StudyDescription = "";

        sc.AutoLoop = 0; // manual looping
        sc.GraphRegion = 0;
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        Input_Enabled.Name = "Enabled";
        Input_Enabled.SetYesNo(false);

        Input_NumBarsToCalculate.Name = "Number of Bars to Calculate";
        Input_NumBarsToCalculate.SetInt(2000);
        Input_NumBarsToCalculate.SetIntLimits(1, MAX_STUDY_LENGTH);

        //Input_DrawStyleOffsetType.Name = "Draw Style Offset Type";
        //Input_DrawStyleOffsetType.SetCustomInputStrings("Percentage;Number of Ticks");

        //Input_PercentageOrTicksOffset.Name = "Percentage or Ticks Offset";
        //Input_PercentageOrTicksOffset.SetFloat(0);

        Input_OrderActionOnAlert.Name = "Order Action on Alert";
        Input_OrderActionOnAlert.SetCustomInputStrings("Buy Entry;Buy Exit;Sell Entry;Sell Exit;Flatten");
        Input_OrderActionOnAlert.SetCustomInputIndex(0);
    }
}
```

```

Input_EvaluateOnBarCloseOnly.Name = "Evaluate on Bar Close Only";
Input_EvaluateOnBarCloseOnly.SetYesNo(true);

Input_SendTradeOrdersToTradeService.Name = "Send Trade Orders to Trade Service";
Input_SendTradeOrdersToTradeService.SetYesNo(false);

Input_MaximumPositionAllowed.Name = "Maximum Position Allowed";
Input_MaximumPositionAllowed.SetInt(1);

Input_MaximumLongTradesPerDay.Name = "Maximum Long Trades Per Day";
Input_MaximumLongTradesPerDay.SetInt(2);

Input_MaximumShortTradesPerDay.Name = "Maximum Short Trades Per Day";
Input_MaximumShortTradesPerDay.SetInt(2);

Input_CancelAllWorkingOrdersOnExit.Name = "Cancel All Working Orders On Exit";
Input_CancelAllWorkingOrdersOnExit.SetYesNo(false);

Input_EnableDebuggingOutput.Name = "Enable Debugging Output";
Input_EnableDebuggingOutput.SetYesNo(false);

Input_AllowTradingOnlyDuringTimeRange.Name = "Allow Trading Only During Time Range";
Input_AllowTradingOnlyDuringTimeRange.SetYesNo(false);

Input_StartTimeForAllowedTimeRange.Name = "Start Time For Allowed Time Range";
Input_StartTimeForAllowedTimeRange.SetTime(HMS_TIME(0, 0, 0));

Input_EndTimeForAllowedTimeRange.Name = "End Time For Allowed Time Range";
Input_EndTimeForAllowedTimeRange.SetTime(HMS_TIME(23, 59, 59));

Input_AllowOnlyOneTradePerBar.Name = "Allow Only One Trade per Bar";
Input_AllowOnlyOneTradePerBar.SetYesNo(true);

Input_SupportReversals.Name = "Support Reversals";
Input_SupportReversals.SetYesNo(false);

Input_AllowMultipleEntriesInSameDirection.Name = "Allow Multiple Entries In Same Direction";
Input_AllowMultipleEntriesInSameDirection.SetYesNo(false);

Input_AllowEntryWithWorkingOrders.Name = "Allow Entry With Working Orders";
Input_AllowEntryWithWorkingOrders.SetYesNo(false);

Input_ControlBarButtonNumberForEnableDisable.Name = "ACS Control Bar Button # for Enable/Disable";
Input_ControlBarButtonNumberForEnableDisable.SetInt(1);
Input_ControlBarButtonNumberForEnableDisable.SetIntLimits(1, MAX_ACS_CONTROL_BAR_BUTTONS);

Input_CancelAllOrdersOnEntries.Name = "Cancel All Orders on Entries";
Input_CancelAllOrdersOnEntries.SetYesNo(false);

Input_CancelAllOrdersOnReversals.Name = "Cancel All Orders on Reversals";
Input_CancelAllOrdersOnReversals.SetYesNo(false);

Input_Version.SetInt(2);

Subgraph_BuyEntry.Name = "Buy";
Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_POINT_ON_HIGH;
Subgraph_BuyEntry.LineWidth = 5;

Subgraph_SellEntry.Name = "Sell";
Subgraph_SellEntry.DrawStyle = DRAWSTYLE_POINT_ON_LOW;
Subgraph_SellEntry.LineWidth = 5;

sc.AllowMultipleEntriesInSameDirection = false;
sc.SupportReversals = false;

```

```

sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;

sc.SupportAttachedOrdersForTrading = false;
sc.UseGUIAttachedOrderSetting = true;
sc.CancelAllOrdersOnEntriesAndReversals = false;
sc.AllowEntryWithWorkingOrders = false;

// Only 1 trade for each Order Action type is allowed per bar.
sc.AllowOnlyOneTradePerBar = true;

//This should be set to true when a trading study uses trading functions.
sc.MaintainTradeStatisticsAndTradesData = true;

return;
}

if (Input_Version.GetInt() < 2)
{
    Input_AllowOnlyOneTradePerBar.SetYesNo(true);
    Input_Version.SetInt(2);
}

sc.AllowOnlyOneTradePerBar = Input_AllowOnlyOneTradePerBar.GetYesNo();
sc.SupportReversals = false;
sc.MaximumPositionAllowed = Input_MaximumPositionAllowed.GetInt();
sc.SendOrdersToTradeService = Input_SendTradeOrdersToTradeService.GetYesNo();
sc.CancelAllWorkingOrdersOnExit = Input_CancelAllWorkingOrdersOnExit.GetYesNo();

sc.CancelAllOrdersOnEntries = Input_CancelAllOrdersOnEntries.GetYesNo();
sc.CancelAllOrdersOnReversals = Input_CancelAllOrdersOnReversals.GetYesNo();

sc.SupportReversals = Input_SupportReversals.GetYesNo();
sc.AllowMultipleEntriesInSameDirection = Input_AllowMultipleEntriesInSameDirection.GetYesNo();
sc.AllowEntryWithWorkingOrders = Input_AllowEntryWithWorkingOrders.GetYesNo();

if (sc.MenuEventID == Input_ControlBarButtonNumberForEnableDisable.GetInt())
{
    const int ButtonState = (sc.PointerEventType == SC_ACS_BUTTON_ON) ? 1 : 0;
    Input_Enabled.SetYesNo(ButtonState);
}

if (!Input_Enabled.GetYesNo() || sc.LastCallToFunction)
    return;

int& r_LastDebugMessageType = sc.GetPersistentInt(1);
int& r_PriorFormulaState = sc.GetPersistentInt(2);

if (sc.IsFullRecalculation)
{
    r_LastDebugMessageType = 0;
    r_PriorFormulaState = 0;

    // set ACS Tool Control Bar tool tip
    sc.SetCustomStudyControlBarButtonHoverText(Input_ControlBarButtonNumberForEnableDisable.GetInt(),
"Enable/Disable Trade System Based on Alert Condition");

    sc.SetCustomStudyControlBarButtonEnable(Input_ControlBarButtonNumberForEnableDisable.GetInt(),
Input_Enabled.GetBoolean());
}

// Figure out the last bar to evaluate
int LastIndex = sc.ArraySize - 1;
if (Input_EvaluateOnBarCloseOnly.GetYesNo())

```

```

--LastIndex;

const int EarliestIndexToCalculate = LastIndex - Input_NumBarsToCalculate.GetInt() + 1;

int CalculationStartIndex = sc.UpdateStartIndex;// sc.GetCalculationStartIndexForStudy();

if (CalculationStartIndex < EarliestIndexToCalculate)
    CalculationStartIndex = EarliestIndexToCalculate;

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;

s_SCPositionData PositionData;

n_ACSIL::s_TradeStatistics TradeStatistics;

SCString TradeNotAllowedReason;

const bool IsDownloadingHistoricalData = sc.ChartIsDownloadingHistoricalData(sc.ChartNumber) != 0;

bool IsTradeAllowed = !sc.IsFullRecalculation && !IsDownloadingHistoricalData;

if (Input_EnableDebuggingOutput.GetYesNo())
{
    if (sc.IsFullRecalculation)
        TradeNotAllowedReason = "Is full recalculation";
    else if (IsDownloadingHistoricalData)
        TradeNotAllowedReason = "Downloading historical data";
}

if (Input_AllowTradingOnlyDuringTimeRange.GetYesNo())
{
    const SCDateTime LastBarDateTime = sc.LatestDateTimeForLastBar;

    const int LastBarTimeInSeconds = LastBarDateTime.GetTimeInSeconds();

    bool LastIndexIsInAllowedTimeRange = false;

    if (Input_StartTimeForAllowedTimeRange.GetTime() <= Input_EndTimeForAllowedTimeRange.GetTime())
    {
        LastIndexIsInAllowedTimeRange = (LastBarTimeInSeconds >= Input_StartTimeForAllowedTimeRange.GetTime()
            && LastBarTimeInSeconds <= Input_EndTimeForAllowedTimeRange.GetTime()
        );
    }
    else //Reversed times.
    {
        LastIndexIsInAllowedTimeRange = (LastBarTimeInSeconds >= Input_StartTimeForAllowedTimeRange.GetTime()
            || LastBarTimeInSeconds <= Input_EndTimeForAllowedTimeRange.GetTime()
        );
    }

    if (!LastIndexIsInAllowedTimeRange)
    {
        IsTradeAllowed = false;
        TradeNotAllowedReason = "Not within allowed time range";
    }
}

bool ParseAndSetFormula = sc.IsFullRecalculation;
SCString DateTimeString;

// Evaluate each of the bars
for (int BarIndex = CalculationStartIndex; BarIndex <= LastIndex; ++BarIndex)
{
    if (Input_EnableDebuggingOutput.GetYesNo())
    {

```

```

        DateTimeString = ". BarDate-Time: ";
        DateTimeString += sc.DateTimeToString(sc.BaseDateTimeIn[BarIndex],
FLAG_DT_COMPLETE_DATETIME_US);
    }

    int FormulaResult = sc.EvaluateAlertConditionFormulaAsBoolean(BarIndex, ParseAndSetFormula);
    ParseAndSetFormula = false;

    bool StateHasChanged = true;
    if (FormulaResult == r_PriorFormulaState)
    {
        TradeNotAllowedReason = "Formula state has not changed. Current state: ";
        TradeNotAllowedReason += FormulaResult ? "True" : "False";
        StateHasChanged = false;
    }

    r_PriorFormulaState = FormulaResult;

    // Determine the result
    if (FormulaResult != 0) //Formula is true
    {
        if (IsTradeAllowed)
        {
            sc.GetTradePosition(PositionData);
            sc.GetTradeStatisticsForSymbolV2(n_ACSIL::STATS_TYPE_DAILY_ALL_TRADES, TradeStatistics);
        }

        if (Input_OrderActionOnAlert.GetIndex() == 0) // Buy entry
        {
            Subgraph_BuyEntry[BarIndex] = 1;
            Subgraph_SellEntry[BarIndex] = 0;

            if (IsTradeAllowed && StateHasChanged)
            {
                if (PositionData.PositionQuantityWithAllWorkingOrders == 0
                    || Input_AllowMultipleEntriesInSameDirection.GetYesNo()
                    || Input_SupportReversals.GetYesNo())
                {
                    if (TradeStatistics.LongTrades < Input_MaximumLongTradesPerDay.GetInt())
                    {
                        s_SCNewOrder NewOrder;
                        NewOrder.OrderQuantity = sc.TradeWindowOrderQuantity;
                        NewOrder.OrderType = SCT_ORDERTYPE_MARKET;

                        int Result = static_cast<int>(sc.BuyEntry(NewOrder, BarIndex));
                        if (Result < 0)
                            sc.AddMessageToTradeServiceLog(sc.GetTradingErrorMessage(Result), false, true);
                    }
                    else if (Input_EnableDebuggingOutput.GetYesNo())
                    {
                        sc.AddMessageToTradeServiceLog("Buy Entry signal ignored. Maximum Long Trades reached.",
false, true);
                    }
                }
            }
            else if (Input_EnableDebuggingOutput.GetYesNo())
            {
                sc.AddMessageToTradeServiceLog("Buy Entry signal ignored. Position exists.", false, true);
            }
        }
        else if (Input_EnableDebuggingOutput.GetYesNo() && r_LastDebugMessageType != 1)
        {
            SCString MessageText("Trading is not allowed. Reason: ");
            MessageText += TradeNotAllowedReason;
            MessageText += DateTimeString;
            sc.AddMessageToTradeServiceLog(MessageText, 0);
        }
    }

```

```

        r_LastDebugMessageType = 1;
    }
}
else if (Input_OrderActionOnAlert.GetIndex() == 1) // Buy exit
{
    Subgraph_BuyEntry[BarIndex] = 0;
    Subgraph_SellEntry[BarIndex] = 1;

    if (IsTradeAllowed && StateHasChanged)
    {
        if (PositionData.PositionQuantity > 0)
        {
            s_SCNewOrder NewOrder;
            NewOrder.OrderQuantity = sc.TradeWindowOrderQuantity;
            NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
            int Result = static_cast<int>(sc.BuyExit(NewOrder, BarIndex));
            if (Result < 0)
                sc.AddMessageToTradeServiceLog(sc.GetTradingErrorTextMessage(Result), false, true);

        }
        else if (Input_EnableDebuggingOutput.GetYesNo())
        {
            sc.AddMessageToTradeServiceLog("Buy Exit signal ignored. Long position does not exist.", false, true);
        }
        else
            r_PriorFormulaState = 0;
    }
    else if (Input_EnableDebuggingOutput.GetYesNo() && r_LastDebugMessageType != 1)
    {
        SCString MessageText("Trading is not allowed. Reason: ");
        MessageText += TradeNotAllowedReason;
        MessageText += DateTimeString;
        sc.AddMessageToTradeServiceLog(MessageText, 0);
        r_LastDebugMessageType = 1;
    }
}
else if (Input_OrderActionOnAlert.GetIndex() == 2) // Sell entry
{
    Subgraph_BuyEntry[BarIndex] = 0;
    Subgraph_SellEntry[BarIndex] = 1;

    if (IsTradeAllowed && StateHasChanged)
    {
        if (PositionData.PositionQuantityWithAllWorkingOrders == 0
            || Input_AllowMultipleEntriesInSameDirection.GetYesNo()
            || Input_SupportReversals.GetYesNo())
        {
            if (TradeStatistics.ShortTrades < Input_MaximumShortTradesPerDay.GetInt())
            {
                s_SCNewOrder NewOrder;
                NewOrder.OrderQuantity = sc.TradeWindowOrderQuantity;
                NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
                int Result = static_cast<int>(sc.SellEntry(NewOrder, BarIndex));
                if (Result < 0)
                    sc.AddMessageToTradeServiceLog(sc.GetTradingErrorTextMessage(Result), false, true);
            }
            else if (Input_EnableDebuggingOutput.GetYesNo())
            {
                sc.AddMessageToTradeServiceLog("Sell Entry signal ignored. Maximum Short Trades reached.",
false, true);
            }
        }
        else if (Input_EnableDebuggingOutput.GetYesNo())
        {
            sc.AddMessageToTradeServiceLog("Sell Entry signal ignored. Position exists.", false, true);
        }
    }
}

```

```

    }
}
else if (Input_EnableDebuggingOutput.GetYesNo() && r_LastDebugMessageType != 1)
{
    SCString MessageText("Trading is not allowed. Reason: ");
    MessageText += TradeNotAllowedReason;
    MessageText += DateTimeString;
    sc.AddMessageToTradeServiceLog(MessageText, 0);
    r_LastDebugMessageType = 1;
}
}
else if (Input_OrderActionOnAlert.GetIndex() == 3) // Sell exit
{
    Subgraph_BuyEntry[BarIndex] = 1;
    Subgraph_SellEntry[BarIndex] = 0;

    if (IsTradeAllowed && StateHasChanged)
    {
        if (PositionData.PositionQuantity < 0)
        {
            s_SCNewOrder NewOrder;
            NewOrder.OrderQuantity = sc.TradeWindowOrderQuantity;
            NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
            int Result = static_cast<int>(sc.SellExit(NewOrder, BarIndex));
            if (Result < 0)
                sc.AddMessageToTradeServiceLog(sc.GetTradingErrorMessage(Result), false, true);
        }
        else if (Input_EnableDebuggingOutput.GetYesNo())
        {
            sc.AddMessageToTradeServiceLog("Sell Exit signal ignored. Short position does not exist.", false, true);
        }
        else
            r_PriorFormulaState = 0;
    }
}
else if (Input_EnableDebuggingOutput.GetYesNo() && r_LastDebugMessageType != 1)
{
    SCString MessageText("Trading is not allowed. Reason: ");
    MessageText += TradeNotAllowedReason;
    MessageText += DateTimeString;
    sc.AddMessageToTradeServiceLog(MessageText, 0);
    r_LastDebugMessageType = 1;
}
}
else if (Input_OrderActionOnAlert.GetIndex() == 4) // Flatten
{
    if (IsTradeAllowed && StateHasChanged)
    {
        int Result = sc.FlattenAndCancelAllOrders();
        if (Result < 0)
        {
            sc.AddMessageToTradeServiceLog(sc.GetTradingErrorMessage(Result), false, true);
        }
    }
}
else if (Input_EnableDebuggingOutput.GetYesNo() && r_LastDebugMessageType != 3)
{
    SCString MessageText("Trading is not allowed. Reason: ");
    MessageText += TradeNotAllowedReason;
    MessageText += DateTimeString;
    sc.AddMessageToTradeServiceLog(MessageText, 0);
    r_LastDebugMessageType = 3;
}
}
}

```

```

else// Formula is not true.
{
    Subgraph_BuyEntry[BarIndex] = 0;
    Subgraph_SellEntry[BarIndex] = 0;

    if (Input_EnableDebuggingOutput.GetYesNo() && r_LastDebugMessageType != 2)
    {
        SCString MessageText("Formula is not true");

        if (!TradeNotAllowedReason.IsEmpty())
        {
            MessageText += ". ";
            MessageText += TradeNotAllowedReason;
        }

        MessageText += DateTimeString;
        sc.AddMessageToTradeServiceLog(MessageText, 0);
        r_LastDebugMessageType = 2;
    }
}

// Reset the prior formula state to 0 if we did not allow trading in order to not prevent trading the next time it is
// allowed if the state did not change.
if (!IsTradeAllowed)
    r_PriorFormulaState = 0;

}

if (sc.IsFullRecalculation)
    r_LastDebugMessageType = 0;
}

```