

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

```
SCSFExport scsf_TradeManagementByStudy(SCStudyInterfaceRef sc)
```

```
{
    // Setup References
    SCInputRef Input_Enabled = sc.Input[0];
    SCInputRef Input_OrderType = sc.Input[1];
    SCInputRef Input_ControlSubgraphRef = sc.Input[2];
    SCInputRef Input_AdjustmentTiming = sc.Input[3];
    SCInputRef Input_AdjustmentFrequency = sc.Input[4];
    SCInputRef Input_TrailingOffset = sc.Input[5];
    SCInputRef Input_ACSButtonNumber = sc.Input[6];
    SCInputRef Input_PositionType = sc.Input[7];
    SCInputRef Input_DetailedLogging = sc.Input[8];
    SCInputRef Input_OnlyModifyInOneDirection = sc.Input[9];
    SCInputRef Input_Version = sc.Input[10];
    SCInputRef Input_AllowZeroValues = sc.Input[11];
    SCInputRef Input_AllowNegativeValues = sc.Input[12];
    SCInputRef Input_AllowCustomPropertiesForControlBarButton = sc.Input[13];
    SCInputRef Input_AttachedOrderToManage = sc.Input[14];
    SCInputRef Input_ChangeToOffAllOtherCustomStudyButtonsOnEnable = sc.Input[15];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Trade Management by Study";
        sc.AutoLoop = 0; // We are using Manual looping for efficiency since historical data is not relevant with this study
        sc.GraphRegion = 0;
        sc.CalculationPrecedence = VERY_LOW_PREC_LEVEL;

        Input_Enabled.Name = "Enabled";
        Input_Enabled.SetYesNo(0);

        Input_OrderType.Name = "Order Type To Manage";
        Input_OrderType.SetCustomInputStrings("Stops;Targets");
        Input_OrderType.SetCustomInputIndex(0);

        Input_ControlSubgraphRef.Name = "Controlling Subgraph Reference";
        Input_ControlSubgraphRef.SetStudySubgraphValues(0, 0);

        Input_AdjustmentTiming.Name = "Order Adjustment Timing";
        Input_AdjustmentTiming.SetCustomInputStrings("Every N Seconds;On Bar Close");
        Input_AdjustmentTiming.SetCustomInputIndex(1);

        Input_AdjustmentFrequency.Name = "Adjustment Frequency (in Seconds)";
        Input_AdjustmentFrequency.SetInt(5);

        Input_TrailingOffset.Name = "Trailing Offset (in Ticks)";
        Input_TrailingOffset.SetInt(2);

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

        Input_PositionType.Name = "Position Type";
        Input_PositionType.SetCustomInputStrings("All Positions;Long Only;Short Only");
        Input_PositionType.SetCustomInputIndex(0);

        Input_DetailedLogging.Name = "Detailed Logging (for debugging)";
        Input_DetailedLogging.SetYesNo(0);

        Input_OnlyModifyInOneDirection.Name = "Only Modify Stops In One Direction";
        Input_OnlyModifyInOneDirection.SetYesNo(false);

        Input_Version.SetInt(2);
    }
}
```

```

Input_AllowZeroValues.Name = "Allow Zero Values";
Input_AllowZeroValues.SetYesNo(false);

Input_AllowNegativeValues.Name = "Allow Negative Values";
Input_AllowNegativeValues.SetYesNo(false);

Input_AllowCustomPropertiesForControlBarButton.Name = "Allow Custom 'Properties' for Control Bar Button";
Input_AllowCustomPropertiesForControlBarButton.SetYesNo(false);

Input_AttachedOrderToManage.Name = "Attached Order to Manage";
Input_AttachedOrderToManage.SetCustomInputStrings("Nearest;OCO Group 1;OCO Group 2;OCO Group 3;OCO
Group 4;OCO Group 5;OCO Group 6;OCO Group 7;OCO Group 8");
Input_AttachedOrderToManage.SetCustomInputIndex(0);

Input_ChangeToOffAllOtherCustomStudyButtonsOnEnable.Name = "Change To Off All Other Custom Study
Buttons On Enable";
Input_ChangeToOffAllOtherCustomStudyButtonsOnEnable.SetYesNo(false);

// Do not receive mouse pointer events when ACS Tool Control Bar button is pressed in/enabled
sc.ReceivePointerEvents = false;

//sc.CancelAllOrdersOnEntries = false;

return;
}

//References to persistent variables
SCDateTime& LastAdjustmentDateTime = sc.GetPersistentSCDateTime(1);

int& MenuID = sc.GetPersistentInt(1);

int & LastLogMessageIdentifier =sc.GetPersistentInt(2);

// Handle pointer events

if (sc.IsFullRecalculation)//Is full recalculation
{
    // set ACS Tool Control Bar tool tip
    sc.SetCustomStudyControlBarButtonHoverText(Input_ACSButtonNumber.GetInt(), "Enable/Disable Trade
Management by Study");

    // set Custom Study Control Bar button text if the input to allow custom properties is not true. Otherwise, rely upon
what the user sets.
    if (!Input_AllowCustomPropertiesForControlBarButton.GetYesNo())
    {
        sc.SetCustomStudyControlBarButtonShortCaption(Input_ACSButtonNumber.GetInt(), "TradeMan\nStudy");
    }

    sc.SetCustomStudyControlBarButtonEnable(Input_ACSButtonNumber.GetInt(), Input_Enabled.GetBoolean() );

    LastLogMessageIdentifier = 0;
}

if (Input_ChangeToOffAllOtherCustomStudyButtonsOnEnable.GetYesNo())
{
    if (sc.MenuEventID >= ACS_BUTTON_1 && sc.MenuEventID <= ACS_BUTTON_150)
    {
        sc.SetCustomStudyControlBarButtonEnable(sc.PriorSelectedCustomStudyControlBarButtonNumber, false);
    }
}

```

```

// Wait for an event to enable study if not already enabled
if (sc.MenuEventID == Input_ACSButtonNumber.GetInt())
{
    const int ButtonState = (sc.PointerEventType == SC_ACS_BUTTON_ON) ? 1 : 0;
    Input_Enabled.SetYesNo(ButtonState);
}

// Execute the order management

// If study is not enabled, exit without doing anything
if (Input_Enabled.GetYesNo() == 0)
{
    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 1)
    {
        LastLogMessageIdentifier = 1;
        sc.AddMessageToLog("Study not enabled.", 0);
    }

    return;
}

//For safety we must never do any order management while historical data is being downloaded.
if (sc.ChartIsDownloadingHistoricalData(sc.ChartNumber))
{
    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 2)
    {
        LastLogMessageIdentifier = 2;
        sc.AddMessageToLog("No order adjustments performed because historical data is being downloaded for chart.",
0);
    }
    return;
}

sc.SendOrdersToTradeService = !sc.GlobalTradeSimulationIsOn;

// Flat state is a good time to reset LastAdjustmentDateTime
s_SCPositionData PositionData;
sc.GetTradePosition (PositionData);

if (PositionData.PositionQuantity == 0)
{
    LastAdjustmentDateTime.Clear();

    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 3)
    {
        LastLogMessageIdentifier = 3;
        sc.AddMessageToLog("Position does not exist.", 0);
    }

    return;
}

if (Input_PositionType.GetIndex() == 1 && PositionData.PositionQuantity < 0)
{
    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 4)
    {
        LastLogMessageIdentifier = 4;
        SCString LogText("Position Type input is set to 'Long Only' and Position is Short. Position: ");
        SCString QuantityString;
        sc.OrderQuantityToString(PositionData.PositionQuantity, QuantityString);
        LogText += QuantityString;
        sc.AddMessageToLog(LogText, 0);
    }
}

```

```

    return;
}

if (Input_PositionType.GetIndex() == 2 && PositionData.PositionQuantity > 0)
{
    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 5)
    {
        LastLogMessageIdentifier = 5;
        SCString LogText("Position Type input is set to 'Short Only' and Position is Long. Position: ");
        SCString QuantityString;
        sc.OrderQuantityToString(PositionData.PositionQuantity, QuantityString);
        LogText += QuantityString;
        sc.AddMessageToLog(LogText, 0);
    }

    return;
}

// If the timing is every N seconds, check that we have surpassed the set frequency
if (Input_AdjustmentTiming.GetInt() == 0 && (sc.LatestDateTimeForLastBar - LastAdjustmentDateTime) <
SCDateTime::SECONDS_Fraction(Input_AdjustmentFrequency.GetFloat()))
{
    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 6)
    {
        LastLogMessageIdentifier = 6;
        sc.AddMessageToLog("Time interval has not elapsed yet for 'Every N Seconds'.", 0);
    }
    return;
}

// Update the LastAdjustmentDateTime

LastAdjustmentDateTime = sc.LatestDateTimeForLastBar;

// Retrieve the controlling subgraph array
SCFloatArray ControllingSubgraph;
sc.GetStudyArrayUsingID(Input_ControlSubgraphRef.GetStudyID(), Input_ControlSubgraphRef.GetSubgraphIndex(),
ControllingSubgraph);

// Based on Input settings, retrieve nearest stop/target order
s_SCTradeOrder ExistingTargetOrStopOrder;
int Result = 0;

if (Input_OrderType.IntValue == 0)
{
    if (Input_AttachedOrderToManage.GetIndex() == 0)
    {
        Result = sc.GetNearestStopOrder(ExistingTargetOrStopOrder);
    }
    else
    {
        Result = sc.GetStopOrderInOCOGroupNumber(Input_AttachedOrderToManage.GetIndex(),
ExistingTargetOrStopOrder);
    }
}
else
{
    if (Input_AttachedOrderToManage.GetIndex() == 0)
    {
        Result = sc.GetNearestTargetOrder(ExistingTargetOrStopOrder);
    }
}

```

```

else
{
    Result = sc.GetTargetOrderInOCOGroupNumber(Input_AttachedOrderToManage.GetIndex(),
ExistingTargetOrStopOrder);
}
}

if (Result == SCTRADING_ORDER_ERROR)
{
    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 7)
    {
        LastLogMessageIdentifier = 7;
        sc.AddMessageToLog("There is no Stop/Target Attached Order found.", 0);
    }
    return;
}

// only process open orders
if (ExistingTargetOrStopOrder.OrderStatusCode != SCT_OSC_OPEN
    && ExistingTargetOrStopOrder.OrderStatusCode != SCT_OSC_HELD)
{
    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 8)
    {
        LastLogMessageIdentifier = 8;
        sc.AddMessageToLog("The found Stop/Target order is not in an Open/Held state.", 0);
    }
    return;
}

// Modify the order
int BarIndex = sc.ArraySize - 1;

const bool AdjustmentTimingIsOnBarClose = Input_AdjustmentTiming.GetIndex() == 1;
if(AdjustmentTimingIsOnBarClose)//On Bar Close
{
    BarIndex = sc.ArraySize - 2;
}

if (!Input_AllowZeroValues.GetYesNo() && ControllingSubgraph[BarIndex] == 0.0)
{
    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 9)
    {
        LastLogMessageIdentifier = 9;
        if (AdjustmentTimingIsOnBarClose)
        {
            sc.AddMessageToLog("The 'Controlling Subgraph Reference' value is 0 at the bar prior to the last bar.", 0);
        }
        else
        {
            sc.AddMessageToLog("The 'Controlling Subgraph Reference' value is 0 at the last bar.", 0);
        }
    }
    return;
}

if (!Input_AllowNegativeValues.GetYesNo() && ControllingSubgraph[BarIndex] < 0)
{
    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 15)
    {
        LastLogMessageIdentifier = 15;
        sc.AddMessageToLog("The 'Controlling Subgraph Reference' value is < 0 at the last bar.", 0);
    }
    return;
}

```

```

}

double NewPrice = 0.0;

if (ExistingTargetOrStopOrder.BuySell == BSE_SELL)
{
    NewPrice = ControllingSubgraph[BarIndex] - Input_TrailingOffset.GetInt() * sc.TickSize;
}
else if (ExistingTargetOrStopOrder.BuySell == BSE_BUY)
{
    NewPrice = ControllingSubgraph[BarIndex] + Input_TrailingOffset.GetInt() * sc.TickSize;
}
else
    return;//Should never happen so no need for error logging.

//Round the price to an actual order price.
NewPrice= sc.RoundToTickSize(NewPrice);

//If the price has not changed, then do not modify.
if (sc.FormattedEvaluateUsingDoubles(ExistingTargetOrStopOrder.Price1, sc.BaseGraphValueFormat,
EQUAL_OPERATOR, NewPrice, sc.BaseGraphValueFormat )
    || sc.FormattedEvaluateUsingDoubles(ExistingTargetOrStopOrder.LastModifyPrice1, sc.BaseGraphValueFormat,
EQUAL_OPERATOR, NewPrice, sc.BaseGraphValueFormat)
    )
{
    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 10)
    {
        LastLogMessageIdentifier = 10;

        SCString LogMessage;
        LogMessage.Format("There is not a new price to modify the order to. Existing order: Price1=%f,
LastModifyPrice1 =%f. New study subgraph price with offset=%f.", ExistingTargetOrStopOrder.Price1,
ExistingTargetOrStopOrder.LastModifyPrice1, NewPrice);

        sc.AddMessageToLog(LogMessage, 0);
    }
    return;
}

// If modify stop price is in wrong direction, then do not modify.
if (Input_OrderType.IntValue == 0 //Stops
    && Input_OnlyModifyInOneDirection.GetYesNo() &&
    ((PositionData.PositionQuantity < 0 && sc.FormattedEvaluateUsingDoubles(NewPrice, sc.BaseGraphValueFormat,
GREATER_OPERATOR, ExistingTargetOrStopOrder.Price1, sc.BaseGraphValueFormat)) ||
    (PositionData.PositionQuantity > 0 && sc.FormattedEvaluateUsingDoubles(NewPrice, sc.BaseGraphValueFormat,
LESS_OPERATOR, ExistingTargetOrStopOrder.Price1, sc.BaseGraphValueFormat)) ) )
{
    if (Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 14)
    {
        LastLogMessageIdentifier = 14;
        sc.AddMessageToLog("Modify stop price is in wrong direction.", 0);
    }
    return;
}

s_SCNewOrder ModifyOrder;
ModifyOrder.InternalOrderID = ExistingTargetOrStopOrder.InternalOrderID;
ModifyOrder.Price1 = NewPrice;

Result = sc.ModifyOrder(ModifyOrder);

if (Result == 1 && Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 11)
{
    LastLogMessageIdentifier = 11;
}

```

```

        SCString LogText;
        LogText.Format("The Stop/Target order has been modified. Internal Order ID: %d. New Price: %f",
ExistingTargetOrStopOrder.InternalOrderID, NewPrice);

        sc.AddMessageToLog( LogText , 0);
    }
    else if (Result == SCT_SKIPPED_AUTO_TRADING_DISABLED && Input_DetailedLogging.GetYesNo() &&
LastLogMessageIdentifier != 13)
    {
        LastLogMessageIdentifier = 13;
        SCString LogText;
        LogText.Format("Trade >> Auto Trading Enabled' is disabled.");

        sc.AddMessageToLog( LogText , 0);
    }
    else if (Result != 1 && Input_DetailedLogging.GetYesNo() && LastLogMessageIdentifier != 12)
    {
        LastLogMessageIdentifier = 12;
        SCString LogText;
        LogText.Format("There was an error modifying the Stop/Target order. Internal Order ID: %d. New Price: %f. Error
Code %d.", ExistingTargetOrStopOrder.InternalOrderID, NewPrice, Result);

        sc.AddMessageToLog( LogText , 0);
    }
}

```