

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

```
typedef std::vector<float> t_FloatVector;
```

```
void TradingSpreadOrderEntry_DrawStatusText(SCStudyInterfaceRef sc);
```

```
void TradingSpreadOrderEntry_UpdateTargetStopSubgraphs(SCStudyInterfaceRef sc, int BeginIndex, int EndIndex);
```

```
/*=====*/
```

```
SCSFExport scsf_TradingSpreadOrderEntry(SCStudyInterfaceRef sc)
```

```
{  
    enum StudyStateEnum : int32_t
```

```
{  
        STATE_UNSET = 0  
        , STATE_DISABLED = 1  
        , STATE_MONITORING_FOR_SPREAD_PRICE_TRIGGER = 2  
        , STATE_MONITORING_FOR_TARGET_AND_STOP = 3  
        , STATE_DISABLED_AFTER_ORDERS_SENT = 4  
        , STATE_DISABLED_INVALID_ENTRY_PRICE = 5  
    };
```

```
    enum SpreadComparisonPriceOperatorEnum : int32_t  
    {  
        LESS_EQUAL = 0  
        , GREATER_EQUAL = 1  
    };
```

```
    SCSubgraphRef Subgraph_SpreadOrderPrice = sc.Subgraph[0];  
    SCSubgraphRef Subgraph_OnChartTextDisplay = sc.Subgraph[1];  
    SCSubgraphRef Subgraph_EntryPriceLine = sc.Subgraph[2];  
    SCSubgraphRef Subgraph_TargetPriceLine = sc.Subgraph[3];  
    SCSubgraphRef Subgraph_StopPriceLine = sc.Subgraph[4];
```

```
    SCInputRef Input_IsActiveAndWaitingForSpreadTriggerPrice = sc.Input[0];  
    SCInputRef Input_Symbol1Symbol = sc.Input[1];  
    SCInputRef Input_Symbol1Side = sc.Input[2];  
    SCInputRef Input_Symbol1Quantity = sc.Input[3];
```

```
    SCInputRef Input_Symbol2Symbol = sc.Input[4];  
    SCInputRef Input_Symbol2Side = sc.Input[5];  
    SCInputRef Input_Symbol2Quantity = sc.Input[6];
```

```
    SCInputRef Input_Symbol3Symbol = sc.Input[7];  
    SCInputRef Input_Symbol3Side = sc.Input[8];  
    SCInputRef Input_Symbol3Quantity = sc.Input[9];
```

```
    SCInputRef Input_Symbol4Symbol = sc.Input[10];  
    SCInputRef Input_Symbol4Side = sc.Input[11];  
    SCInputRef Input_Symbol4Quantity = sc.Input[12];
```

```
    SCInputRef Input_SpreadEntryPrice = sc.Input[13];  
    SCInputRef Input_SpreadEntryPriceOperator = sc.Input[14];
```

```
    SCInputRef Input_NumberOfBarsForSubgraphs = sc.Input[15];  
    SCInputRef Input_DrawZeroValues = sc.Input[16];
```

```
    SCInputRef Input_UseTargetAfterFillOfSpreadEntry = sc.Input[17];  
    SCInputRef Input_TargetSpreadPrice = sc.Input[18];
```

```
    SCInputRef Input_UseStopAfterFillOfSpreadEntry = sc.Input[19];  
    SCInputRef Input_StopSpreadPrice = sc.Input[20];
```

```
    SCInputRef Input_DisableIfImmediateFillOfSpreadAfterSpreadPriceSet = sc.Input[21];  
    SCInputRef Input_UseFullContractValuesForSpreadPrices = sc.Input[22];
```

```

if (sc.SetDefaults)
{
    sc.GraphName = "Trading: Spread Order Entry";
    sc.AutoLoop = 0;//Manual looping
    sc.GraphRegion = 1;
    sc.ScaleRangeType= SCALE_AUTO;
    sc.ValueFormat = VALUEFORMAT_INHERITED;
    sc.UpdateAlways = 1;
    sc.ReceivePointerEvents = ACS_RECEIVE_POINTER_EVENTS_ALWAYS;

    Subgraph_SpreadOrderPrice.Name = "Spread Price";
    Subgraph_SpreadOrderPrice.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_SpreadOrderPrice.PrimaryColor = RGB(0,128,255);
    Subgraph_SpreadOrderPrice.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_VALUES_SCALE |
LL_VALUE_ALIGN_CENTER | LL_DISPLAY_NAME | LL_NAME_ALIGN_LEFT_EDGE | LL_NAME_ALIGN_CENTER;
    Subgraph_SpreadOrderPrice.DrawZeros = true;

    Subgraph_OnChartTextDisplay.Name = "Chart Text Display";
    Subgraph_OnChartTextDisplay.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
    Subgraph_OnChartTextDisplay.PrimaryColor = RGB(255, 165, 0);
    Subgraph_OnChartTextDisplay.LineWidth = 10;

    Subgraph_EntryPriceLine.Name = "Entry Price";
    Subgraph_EntryPriceLine.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_EntryPriceLine.LineWidth = 2;
    Subgraph_EntryPriceLine.PrimaryColor = RGB(0, 255, 128);
    Subgraph_EntryPriceLine.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_VALUES_SCALE |
LL_VALUE_ALIGN_CENTER | LL_DISPLAY_NAME | LL_NAME_ALIGN_LEFT_EDGE | LL_NAME_ALIGN_CENTER;
    Subgraph_EntryPriceLine.DrawZeros = true;

    Subgraph_TargetPriceLine.Name = "Target Price";
    Subgraph_TargetPriceLine.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_TargetPriceLine.LineWidth = 2;
    Subgraph_TargetPriceLine.PrimaryColor = RGB(255, 100, 0);
    Subgraph_TargetPriceLine.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_VALUES_SCALE |
LL_VALUE_ALIGN_CENTER | LL_DISPLAY_NAME | LL_NAME_ALIGN_LEFT_EDGE | LL_NAME_ALIGN_CENTER;
    Subgraph_TargetPriceLine.DrawZeros = true;

    Subgraph_StopPriceLine.Name = "Stop Price";
    Subgraph_StopPriceLine.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_StopPriceLine.LineWidth = 2;
    Subgraph_StopPriceLine.PrimaryColor = RGB(255, 100, 0);
    Subgraph_StopPriceLine.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_VALUES_SCALE |
LL_VALUE_ALIGN_CENTER | LL_DISPLAY_NAME | LL_NAME_ALIGN_LEFT_EDGE | LL_NAME_ALIGN_CENTER;
    Subgraph_StopPriceLine.DrawZeros = true;

    Input_IsActiveAndWaitingForSpreadTriggerPrice.Name = "Is Active and Waiting for Spread Trigger Price";
    Input_IsActiveAndWaitingForSpreadTriggerPrice.SetYesNo(false);

    Input_Symbol1Symbol.Name = "Leg 1 Symbol";
    Input_Symbol1Symbol.SetSymbol("");

    Input_Symbol1Side.Name = "Leg 1 Side";
    Input_Symbol1Side.SetCustomInputStrings("Buy;Sell");
    Input_Symbol1Side.SetCustomInputIndex(0);

    Input_Symbol1Quantity.Name = "Leg 1 Quantity";
    Input_Symbol1Quantity.SetInt(1);

    Input_Symbol2Symbol.Name = "Leg 2 Symbol";
    Input_Symbol2Symbol.SetSymbol("");

    Input_Symbol2Side.Name = "Leg 2 Side";
    Input_Symbol2Side.SetCustomInputStrings("Buy;Sell");
    Input_Symbol2Side.SetCustomInputIndex(1);

```

```

Input_Symbol2Quantity.Name = "Leg 2 Quantity";
Input_Symbol2Quantity.SetInt(1);

Input_Symbol3Symbol.Name = "Leg 3 Symbol";
Input_Symbol3Symbol.SetSymbol("");

Input_Symbol3Side.Name = "Leg 3 Side";
Input_Symbol3Side.SetCustomInputStrings("Buy;Sell");
Input_Symbol3Side.SetCustomInputIndex(0);

Input_Symbol3Quantity.Name = "Leg 3 Quantity";
Input_Symbol3Quantity.SetInt(0);

Input_Symbol4Symbol.Name = "Leg 4 Symbol";
Input_Symbol4Symbol.SetSymbol("");

Input_Symbol4Side.Name = "Leg 4 Side";
Input_Symbol4Side.SetCustomInputStrings("Buy;Sell");
Input_Symbol4Side.SetCustomInputIndex(0);

Input_Symbol4Quantity.Name = "Leg 4 Quantity";
Input_Symbol4Quantity.SetInt(0);

Input_SpreadEntryPrice.Name = "Spread Entry Price";
Input_SpreadEntryPrice.SetFloat(0);

Input_SpreadEntryPriceOperator.Name = "Spread Entry Price Operator";
Input_SpreadEntryPriceOperator.SetCustomInputStrings("Less or Equal;Greater or Equal");
Input_SpreadEntryPriceOperator.SetCustomInputIndex(LESS_EQUAL);

Input_NumberOfBarsForSubgraphs.Name = "Number of Bars for Subgraphs";
Input_NumberOfBarsForSubgraphs.SetInt(200);
Input_NumberOfBarsForSubgraphs.SetIntLimits(1, 2000);

Input_DrawZeroValues.Name = "Draw Zero Values";
Input_DrawZeroValues.SetYesNo(false);

Input_UseTargetAfterFillOfSpreadEntry.Name = "Use Target after Fill of Spread Entry";
Input_UseTargetAfterFillOfSpreadEntry.SetYesNo(false);

Input_TargetSpreadPrice.Name = "Target Spread Price";
Input_TargetSpreadPrice.SetFloat(0);

Input_UseStopAfterFillOfSpreadEntry.Name = "Use Stop after Fill of Spread Entry";
Input_UseStopAfterFillOfSpreadEntry.SetYesNo(false);

Input_StopSpreadPrice.Name = "Stop Spread Price";
Input_StopSpreadPrice.SetFloat(0);

Input_DisableIfImmediateFillOfSpreadAfterSpreadPriceSet.Name = "Disable if Immediate Fill of Spread After Spread
Price Set";
Input_DisableIfImmediateFillOfSpreadAfterSpreadPriceSet.SetYesNo(false);

Input_UseFullContractValuesForSpreadPrices.Name = "Use Full Contract Values for Spread Prices";
Input_UseFullContractValuesForSpreadPrices.SetYesNo(false);

sc.AllowMultipleEntriesInSameDirection = 1;
sc.AllowOppositeEntryWithOpposingPositionOrOrders = 1;
sc.CancelAllOrdersOnEntriesAndReversals = 0;
sc.AllowOnlyOneTradePerBar = 0;
sc.SupportReversals = 0;
sc.AllowEntryWithWorkingOrders = 1;
sc.SupportAttachedOrdersForTrading = 0;
sc.UseGUIAttachedOrderSetting = 0;

```

```

sc.MaximumPositionAllowed = 100000000;

return;
}

sc.DisplayAsMainPriceGraph = false;

int& r_EnableSpreadOrderEntryMenuID = sc.GetPersistentInt(1);
int& r_DisableSpreadOrderEntryMenuID = sc.GetPersistentInt(2);
int& r_DisabledActiveStatus = sc.GetPersistentInt(3);
int& r_CurrentState = sc.GetPersistentInt(4);
int& r_DrawingNumber = sc.GetPersistentInt(5);
int& r_OrderModifyMode = sc.GetPersistentInt(6);

//will both be 0 upon opening the Chartbook or first-time study added
double& r_PriorSpreadEntryPrice = sc.GetPersistentDouble(1);
int& r_SpreadEntryPriceHasChanged = sc.GetPersistentInt(7);

SCString& r_CurrentOnScreenMessage = sc.GetPersistentSCString(1);

const int PersistentHistoricalSpreadDataKey = 1;
t_FloatVector* p_PersistentHistoricalSpreadData = reinterpret_cast<t_FloatVector*>
(sc.GetPersistentPointer(PersistentHistoricalSpreadDataKey));

if (sc.LastCallToFunction)
{
    // Be sure to remove the menu commands when study is removed
    sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_EnableSpreadOrderEntryMenuID);
    sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_DisableSpreadOrderEntryMenuID);

    if (p_PersistentHistoricalSpreadData != NULL)
    {
        delete p_PersistentHistoricalSpreadData;
        sc.SetPersistentPointer(PersistentHistoricalSpreadDataKey, NULL);
    }

    return;
}

if (p_PersistentHistoricalSpreadData == NULL)
{
    p_PersistentHistoricalSpreadData = new t_FloatVector;

    if (p_PersistentHistoricalSpreadData != NULL)
        sc.SetPersistentPointer(PersistentHistoricalSpreadDataKey, p_PersistentHistoricalSpreadData);
}

if (p_PersistentHistoricalSpreadData == NULL)
    return;

t_FloatVector& r_PersistentHistoricalSpreadData = *p_PersistentHistoricalSpreadData;
r_PersistentHistoricalSpreadData.resize(Input_NumberOfBarsForSubgraphs.GetInt());

// this will be true when study is added to chart, and when Chartbook is opened with study on chart. This is done as a
safety.
if (r_DisabledActiveStatus == 0)
{
    Input_IsActiveAndWaitingForSpreadTriggerPrice.SetYesNo(false);
    r_CurrentState = STATE_DISABLED;
    r_DisabledActiveStatus = 1;
}

```

```

sc.SendOrdersToTradeService = sc.GlobalTradeSimulationIsOn ? 0 : 1;

//sc.SendOrdersToTradeService = 0;// TEMPORARY: Safety during development and testing

if (sc.IsFullRecalculation)
{
    r_DrawingNumber = 0;

    if (Input_NumberOfBarsForSubgraphs.GetInt() == 0)
        Input_NumberOfBarsForSubgraphs.SetInt(100);

    Subgraph_SpreadOrderPrice.DrawZeros = Input_DrawZeroValues.GetBoolean();
    Subgraph_EntryPriceLine.DrawZeros = Input_DrawZeroValues.GetBoolean();

    if (Input_UseTargetAfterFillOfSpreadEntry.GetYesNo())
        Subgraph_TargetPriceLine.DrawZeros = Input_DrawZeroValues.GetBoolean();
    else
        Subgraph_TargetPriceLine.DrawZeros = false;

    if (Input_UseStopAfterFillOfSpreadEntry.GetYesNo())
        Subgraph_StopPriceLine.DrawZeros = Input_DrawZeroValues.GetBoolean();
    else
        Subgraph_StopPriceLine.DrawZeros = false;

    if (r_EnableSpreadOrderEntryMenuID <= 0)
    {
        r_EnableSpreadOrderEntryMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Enable Spread
Order Entry");
    }

    if (r_DisableSpreadOrderEntryMenuID <= 0)
    {
        r_DisableSpreadOrderEntryMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Disable Spread
Order Entry");
    }

    if (r_CurrentState == STATE_UNSET)
        r_CurrentState = STATE_DISABLED;
}

if (Input_IsActiveAndWaitingForSpreadTriggerPrice.GetYesNo())
{
    sc.SetACSCChartShortcutMenuItemEnabled(sc.ChartNumber, r_EnableSpreadOrderEntryMenuID, 0);
    sc.SetACSCChartShortcutMenuItemEnabled(sc.ChartNumber, r_DisableSpreadOrderEntryMenuID, 1);
}
else
{
    sc.SetACSCChartShortcutMenuItemEnabled(sc.ChartNumber, r_EnableSpreadOrderEntryMenuID, 1);
    sc.SetACSCChartShortcutMenuItemEnabled(sc.ChartNumber, r_DisableSpreadOrderEntryMenuID, 0);
}

if (!sc.ChartTradeModeEnabled)
{
    r_CurrentOnScreenMessage = "Chart Trade Mode is not Enabled";
    TradingSpreadOrderEntry_DrawStatusText(sc);
    return;
}

// handle set trigger price menu clicks
if (sc.MenuEventID != 0 && sc.MenuEventID == r_EnableSpreadOrderEntryMenuID)
{
    Input_IsActiveAndWaitingForSpreadTriggerPrice.SetYesNo(true);

    if (r_CurrentState == STATE_DISABLED

```

```

    || r_CurrentState == STATE_DISABLED_AFTER_ORDERS_SENT
    || r_CurrentState == STATE_DISABLED_INVALID_ENTRY_PRICE)
{
    r_CurrentState = STATE_MONITORING_FOR_SPREAD_PRICE_TRIGGER;
    r_CurrentOnScreenMessage = "Monitoring for spread entry price";
    TradingSpreadOrderEntry_DrawStatusText(sc);
}

sc.SetACSCChartShortcutMenuItemEnabled(sc.ChartNumber, r_EnableSpreadOrderEntryMenuID, 0);
sc.SetACSCChartShortcutMenuItemEnabled(sc.ChartNumber, r_DisableSpreadOrderEntryMenuID, 1);

}
else if (sc.MenuEventID != 0 && sc.MenuEventID == r_DisableSpreadOrderEntryMenuID)
{
    Input_IsActiveAndWaitingForSpreadTriggerPrice.SetYesNo(false);
    r_CurrentState = STATE_DISABLED;
    r_CurrentOnScreenMessage = "Disabled";
    TradingSpreadOrderEntry_DrawStatusText(sc);

    sc.SetACSCChartShortcutMenuItemEnabled(sc.ChartNumber, r_EnableSpreadOrderEntryMenuID, 1);
    sc.SetACSCChartShortcutMenuItemEnabled(sc.ChartNumber, r_DisableSpreadOrderEntryMenuID, 0);
}

const int NUM_SPREAD_ORDERS = 4;

SCString SpreadSymbols[NUM_SPREAD_ORDERS];
int32_t SpreadSides[NUM_SPREAD_ORDERS];
uint32_t SpreadQuantities[NUM_SPREAD_ORDERS];

SpreadSymbols[0] = Input_Symbol1Symbol.GetSymbol();
SpreadSymbols[1] = Input_Symbol2Symbol.GetSymbol();
SpreadSymbols[2] = Input_Symbol3Symbol.GetSymbol();
SpreadSymbols[3] = Input_Symbol4Symbol.GetSymbol();

SpreadSides[0] = Input_Symbol1Side.GetIndex();
SpreadSides[1] = Input_Symbol2Side.GetIndex();
SpreadSides[2] = Input_Symbol3Side.GetIndex();
SpreadSides[3] = Input_Symbol4Side.GetIndex();

SpreadQuantities[0] = Input_Symbol1Quantity.GetInt();
SpreadQuantities[1] = Input_Symbol2Quantity.GetInt();
SpreadQuantities[2] = Input_Symbol3Quantity.GetInt();
SpreadQuantities[3] = Input_Symbol4Quantity.GetInt();

if (SpreadSymbols[0].IsEmpty() || SpreadSymbols[1].IsEmpty()
    || SpreadQuantities[0] == 0 || SpreadQuantities[1] == 0)
{
    r_CurrentOnScreenMessage = "Settings not complete for spread entry";
    TradingSpreadOrderEntry_DrawStatusText(sc);
    return;
}

const bool UseFullContractValues = Input_UseFullContractValuesForSpreadPrices.GetYesNo();

double CurrentSpreadValue = 0.0;
bool IsSpreadPriceValid = true;

int NumberOfPricesInCalculation = 0;
int NumberOfLegs = 0;
int NumberOfBuys = 0;
int NumberOfSells = 0;

for (int SpreadOrderDataIndex = 0; SpreadOrderDataIndex < NUM_SPREAD_ORDERS; ++SpreadOrderDataIndex )

```

```

{

    if (SpreadQuantities[SpreadOrderDataIndex] == 0.0)
        continue;

    if (SpreadSymbols[SpreadOrderDataIndex].IsEmpty())
        continue;

    ++NumberOfLegs;

    s_SCBasicSymbolData BasicSymbolData;
    sc.GetBasicSymbolData(SpreadSymbols[SpreadOrderDataIndex].GetChars(), BasicSymbolData, true);

    if (BasicSymbolData.Bid == 0.0f || BasicSymbolData.Ask == 0.0f)
    {
        r_CurrentOnScreenMessage = "Bid or Ask prices are zero";
        TradingSpreadOrderEntry_DrawStatusText(sc);
        IsSpreadPriceValid = false;
    }

    if (SpreadSides[SpreadOrderDataIndex] == 0)//Buy
    {
        const float AskPrice = BasicSymbolData.Ask * BasicSymbolData.DisplayPriceMultiplier;
        const double IndividualSymbolValue = AskPrice * SpreadQuantities[SpreadOrderDataIndex];

        if (UseFullContractValues && BasicSymbolData.CurrencyValuePerTick > 0)
        {
            CurrentSpreadValue += (IndividualSymbolValue / BasicSymbolData.TickSize *
BasicSymbolData.CurrencyValuePerTick);
        }
        else
            CurrentSpreadValue += IndividualSymbolValue;

        ++NumberOfPricesInCalculation;
        NumberOfBuys++;
    }
    else if (SpreadSides[SpreadOrderDataIndex] == 1)//Sell
    {
        const float BidPrice = BasicSymbolData.Bid * BasicSymbolData.DisplayPriceMultiplier;
        const double IndividualSymbolValue = BidPrice * SpreadQuantities[SpreadOrderDataIndex];

        if (UseFullContractValues && BasicSymbolData.CurrencyValuePerTick > 0)
        {
            CurrentSpreadValue -= (IndividualSymbolValue / BasicSymbolData.TickSize *
BasicSymbolData.CurrencyValuePerTick);
        }
        else
            CurrentSpreadValue -= IndividualSymbolValue;

        ++NumberOfPricesInCalculation;
        NumberOfSells++;
    }
}

double RegionHigh = 0.0;
double RegionLow = 0.0;

sc.GetGraphVisibleHighAndLow(RegionHigh, RegionLow);

bool AllowModify = !((RegionHigh == 0.0 && RegionLow == 0.0)
|| sc.ActiveToolYValue > RegionHigh
|| sc.ActiveToolYValue < RegionLow);

```

```

if (!AllowModify)
    r_OrderModifyMode = 0;

if (AllowModify && Input_UseTargetAfterFillOfSpreadEntry.GetYesNo()
    && r_OrderModifyMode != 2 && r_OrderModifyMode != 3) //Not modifying Stop or Entry
{
    double TargetPrice = Input_TargetSpreadPrice.GetDouble();
    int YPosition = sc.RegionValueToYPixelCoordinate(static_cast<float>(TargetPrice), sc.GraphRegion);

    if (sc.PointerEventType == SC_POINTER_BUTTON_DOWN)
    {
        if (YPosition >= sc.PointerVertWindowCoord - 5
            && YPosition <= sc.PointerVertWindowCoord + 5)
            r_OrderModifyMode = 1;
    }

    if (sc.PointerEventType == SC_POINTER_BUTTON_UP)
        r_OrderModifyMode = 0;

    if (r_OrderModifyMode == 1 && sc.PointerEventType == SC_POINTER_MOVE)
    {
        Input_TargetSpreadPrice.SetFloat(static_cast<float>(sc.RoundToTickSize(sc.ActiveToolYValue)));
    }
}

if (AllowModify && Input_UseStopAfterFillOfSpreadEntry.GetYesNo()
    && r_OrderModifyMode != 1 && r_OrderModifyMode != 3)
{
    double StopPrice = Input_StopSpreadPrice.GetDouble();
    int Yposition = sc.RegionValueToYPixelCoordinate(static_cast<float>(StopPrice), sc.GraphRegion);

    if (sc.PointerEventType == SC_POINTER_BUTTON_DOWN)
    {
        if (Yposition >= sc.PointerVertWindowCoord - 5
            && Yposition <= sc.PointerVertWindowCoord + 5)
            r_OrderModifyMode = 2;
    }

    if (sc.PointerEventType == SC_POINTER_BUTTON_UP)
        r_OrderModifyMode = 0;

    if (r_OrderModifyMode == 2 && sc.PointerEventType == SC_POINTER_MOVE)
    {
        Input_StopSpreadPrice.SetFloat(static_cast<float>(sc.RoundToTickSize(sc.ActiveToolYValue)));
    }
}

if (AllowModify && r_OrderModifyMode != 1 && r_OrderModifyMode != 2)
{
    double EntryPrice = Input_SpreadEntryPrice.GetDouble();
    int YPosition = sc.RegionValueToYPixelCoordinate(static_cast<float>(EntryPrice), sc.GraphRegion);

    if (sc.PointerEventType == SC_POINTER_BUTTON_DOWN)
    {
        if (YPosition >= sc.PointerVertWindowCoord - 5
            && YPosition <= sc.PointerVertWindowCoord + 5)
            r_OrderModifyMode = 3;
    }

    if (sc.PointerEventType == SC_POINTER_BUTTON_UP)
        r_OrderModifyMode = 0;

    if (r_OrderModifyMode == 3 && sc.PointerEventType == SC_POINTER_MOVE)
    {

```

```

    Input_SpreadEntryPrice.SetFloat(static_cast<float>(sc.RoundToTickSize(sc.ActiveToolYValue)));
}
}

//Check here for changes with spread order entry price
if (r_PriorSpreadEntryPrice != Input_SpreadEntryPrice.GetFloat())
    r_SpreadEntryPriceHasChanged = 1;

r_PriorSpreadEntryPrice = Input_SpreadEntryPrice.GetFloat();

int BarStartIndex = sc.UpdateStartIndex;
int EarliestIndex = sc.ArraySize - Input_NumberOfBarsForSubgraphs.GetInt();
if (EarliestIndex > BarStartIndex)
{
    BarStartIndex = EarliestIndex;
}

if (r_OrderModifyMode != 0 && BarStartIndex > EarliestIndex)
    BarStartIndex = EarliestIndex;

// zero values before the bar starting index for the values
for (int BarIndex = sc.UpdateStartIndex; BarIndex < BarStartIndex; BarIndex++)
{
    Subgraph_EntryPriceLine[BarIndex] = 0;
    Subgraph_TargetPriceLine[BarIndex] = 0;
    Subgraph_StopPriceLine[BarIndex] = 0;
    Subgraph_SpreadOrderPrice[BarIndex] = 0;
}

for (int BarIndex = BarStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    Subgraph_EntryPriceLine[BarIndex] = Input_SpreadEntryPrice.GetFloat();

    if (Input_UseTargetAfterFillOfSpreadEntry.GetYesNo())
        Subgraph_TargetPriceLine[BarIndex] = Input_TargetSpreadPrice.GetFloat();

    if (Input_UseStopAfterFillOfSpreadEntry.GetYesNo())
        Subgraph_StopPriceLine[BarIndex] = Input_StopSpreadPrice.GetFloat();
}

if (sc.IsFullRecalculation)
{
    const int VectorSize = static_cast<int>(r_PersistentHistoricalSpreadData.size());

    int OutputIndex = sc.ArraySize - VectorSize;

    for (int Index = 0; Index < VectorSize; ++Index)
    {
        Subgraph_SpreadOrderPrice[OutputIndex] = r_PersistentHistoricalSpreadData[Index];
        ++OutputIndex;
    }
}
else if (IsSpreadPriceValid
    && Subgraph_SpreadOrderPrice[sc.ArraySize - 1] != static_cast<float>(CurrentSpreadValue))
{
    //shift the current spread price values back
    for (int BarIndex = EarliestIndex; BarIndex < sc.ArraySize - 1; BarIndex++)
    {
        Subgraph_SpreadOrderPrice[BarIndex] = Subgraph_SpreadOrderPrice[BarIndex + 1];
    }

    const int VectorSize = static_cast<int>(r_PersistentHistoricalSpreadData.size());
    for (int Index = 0; Index < VectorSize-1; ++Index)
    {

```

```

        r_PersistentHistoricalSpreadData[Index] = r_PersistentHistoricalSpreadData[Index + 1];
    }
}

if (IsSpreadPriceValid)
{
    Subgraph_SpreadOrderPrice[sc.ArraySize - 1] = static_cast<float>(CurrentSpreadValue);

    if (!r_PersistentHistoricalSpreadData.empty())
        r_PersistentHistoricalSpreadData.back() = static_cast<float>(CurrentSpreadValue);
}

for (int BarIndex = EarliestIndex - 1; BarIndex >= 0; --BarIndex)
{
    if (Subgraph_SpreadOrderPrice[BarIndex] == 0
        && Subgraph_EntryPriceLine[BarIndex] == 0
        && Subgraph_TargetPriceLine[BarIndex] == 0
        && Subgraph_StopPriceLine[BarIndex] == 0)
        break;

    Subgraph_SpreadOrderPrice[BarIndex] = 0;
    Subgraph_EntryPriceLine[BarIndex] = 0;
    Subgraph_TargetPriceLine[BarIndex] = 0;
    Subgraph_StopPriceLine[BarIndex] = 0;
}

if (sc.IsFullRecalculation)//Do not do any spread price evaluation on a full recalculation
{
    if (r_CurrentState == STATE_MONITORING_FOR_SPREAD_PRICE_TRIGGER
        || r_CurrentState == STATE_MONITORING_FOR_TARGET_AND_STOP)
    {
        r_CurrentOnScreenMessage = "Waiting to begin monitoring";
        TradingSpreadOrderEntry_DrawStatusText(sc);
    }

    return;
}

if (NumberOfPricesInCalculation != NumberOfLegs)
{
    r_CurrentOnScreenMessage = "Number of prices in calculation != to the number of legs";
    TradingSpreadOrderEntry_DrawStatusText(sc);
    return;
}

if (NumberOfLegs == 2 && NumberOfBuys != NumberOfSells)
{
    r_CurrentOnScreenMessage = "Disabled: Number of buys != number of sells for 2 legs";
    Input_IsActiveAndWaitingForSpreadTriggerPrice.SetYesNo(false);
    r_CurrentState = STATE_DISABLED;
    TradingSpreadOrderEntry_DrawStatusText(sc);
    return;
}

bool UseTargetOrStop = Input_UseTargetAfterFillOfSpreadEntry.GetYesNo() ||
Input_UseStopAfterFillOfSpreadEntry.GetYesNo();

if (!Input_IsActiveAndWaitingForSpreadTriggerPrice.GetYesNo()
    || (r_CurrentState == STATE_MONITORING_FOR_TARGET_AND_STOP && !UseTargetOrStop))
{
    if (r_CurrentState != STATE_DISABLED

```

```

    && r_CurrentState != STATE_DISABLED_AFTER_ORDERS_SENT
    && r_CurrentState != STATE_DISABLED_INVALID_ENTRY_PRICE)
    r_CurrentState = STATE_DISABLED;

    if(r_CurrentState != STATE_DISABLED_AFTER_ORDERS_SENT
    && r_CurrentState != STATE_DISABLED_INVALID_ENTRY_PRICE)
    r_CurrentOnScreenMessage = "Disabled";

    TradingSpreadOrderEntry_DrawStatusText(sc);
    return;
}

if(r_CurrentState == STATE_DISABLED
    || r_CurrentState == STATE_DISABLED_AFTER_ORDERS_SENT
    || r_CurrentState == STATE_DISABLED_INVALID_ENTRY_PRICE)
{
    r_CurrentState = STATE_MONITORING_FOR_SPREAD_PRICE_TRIGGER;
}

if (r_CurrentState == STATE_MONITORING_FOR_SPREAD_PRICE_TRIGGER)
{
    r_CurrentOnScreenMessage = "Monitoring for spread entry price";
    TradingSpreadOrderEntry_DrawStatusText(sc);
}

if (r_OrderModifyMode != 0)
    return;

if (!IsSpreadPriceValid)
    return;

int NumberOfPricesToCompare = 0;
double SpreadPricesToCompareTo[2] = {};
SpreadComparisonPriceOperatorEnum ComparisonOperators[2] = {LESS_EQUAL};
if (r_CurrentState == STATE_MONITORING_FOR_SPREAD_PRICE_TRIGGER)
{
    SpreadPricesToCompareTo[0] = Input_SpreadEntryPrice.GetFloat();
    NumberOfPricesToCompare = 1;
    ComparisonOperators[0]
        = static_cast<SpreadComparisonPriceOperatorEnum>(Input_SpreadEntryPriceOperator.GetIndex());
}
else if (r_CurrentState == STATE_MONITORING_FOR_TARGET_AND_STOP)
{
    // In the case of greater or equal, invert the operator for the exit order that is lower than the entry.

    // In the case of less or equal, invert the operator for the exit order that is higher than the entry.
    if (Input_UseTargetAfterFillOfSpreadEntry.GetYesNo())
    {
        SpreadPricesToCompareTo[NumberOfPricesToCompare] = Input_TargetSpreadPrice.GetFloat();

        if (Input_SpreadEntryPriceOperator.GetIndex() == LESS_EQUAL
            && SpreadPricesToCompareTo[NumberOfPricesToCompare] > Input_SpreadEntryPrice.GetFloat())
        {
            ComparisonOperators[NumberOfPricesToCompare] = GREATER_EQUAL;
        }
        else if (Input_SpreadEntryPriceOperator.GetIndex() == GREATER_EQUAL
            && SpreadPricesToCompareTo[NumberOfPricesToCompare] < Input_SpreadEntryPrice.GetFloat())
        {
            ComparisonOperators[NumberOfPricesToCompare] = LESS_EQUAL;
        }
    }
    else

```

```

{
    ComparisonOperators[NumberOfPricesToCompare]
        = static_cast<SpreadComparisonPriceOperatorEnum>(Input_SpreadEntryPriceOperator.GetIndex());
}

NumberOfPricesToCompare++;
}

if (Input_UseStopAfterFillOfSpreadEntry.GetYesNo())
{
    SpreadPricesToCompareTo[NumberOfPricesToCompare] = Input_StopSpreadPrice.GetFloat();

    if (Input_SpreadEntryPriceOperator.GetIndex() == LESS_EQUAL
        && SpreadPricesToCompareTo[NumberOfPricesToCompare] > Input_SpreadEntryPrice.GetFloat())
    {
        ComparisonOperators[NumberOfPricesToCompare] = GREATER_EQUAL;
    }
    else if (Input_SpreadEntryPriceOperator.GetIndex() == GREATER_EQUAL
        && SpreadPricesToCompareTo[NumberOfPricesToCompare] < Input_SpreadEntryPrice.GetFloat())
    {
        ComparisonOperators[NumberOfPricesToCompare] = LESS_EQUAL;
    }
    else
    {
        ComparisonOperators[NumberOfPricesToCompare]
            = static_cast<SpreadComparisonPriceOperatorEnum>(Input_SpreadEntryPriceOperator.GetIndex());
    }

    NumberOfPricesToCompare++;
}
}

bool SpreadPriceMet = false;
double SpreadTriggerPrice = 0.0;

for(int Index = 0; Index < NumberOfPricesToCompare; Index++)
{
    if (ComparisonOperators[Index] == LESS_EQUAL)
    {
        if (CurrentSpreadValue <= SpreadPricesToCompareTo[Index])
        {
            if (r_CurrentState == STATE_MONITORING_FOR_SPREAD_PRICE_TRIGGER)
                r_CurrentOnScreenMessage = "Triggered | ";
            else
                r_CurrentOnScreenMessage = "Target/Stop Triggered | ";

            SpreadPriceMet = true;
            SpreadTriggerPrice = CurrentSpreadValue;

            r_CurrentOnScreenMessage += sc.FormatGraphValue(SpreadTriggerPrice, sc.BaseGraphValueFormat);
            r_CurrentOnScreenMessage += " | ";
        }
    }
    else if (ComparisonOperators[Index] == GREATER_EQUAL)
    {
        if (CurrentSpreadValue >= SpreadPricesToCompareTo[Index])
        {
            if (r_CurrentState == STATE_MONITORING_FOR_SPREAD_PRICE_TRIGGER)
                r_CurrentOnScreenMessage = "Triggered | ";
            else
                r_CurrentOnScreenMessage = "Target/Stop Triggered | ";

            SpreadPriceMet = true;

```

```

        SpreadTriggerPrice = CurrentSpreadValue;

        r_CurrentOnScreenMessage += sc.FormatGraphValue(SpreadTriggerPrice, sc.BaseGraphValueFormat);
        r_CurrentOnScreenMessage += " | ";
    }

}
else
    return;

}

if (SpreadPriceMet)
{
    if (r_SpreadEntryPriceHasChanged
        && Input_DisableIfImmediateFillOfSpreadAfterSpreadPriceSet.GetYesNo())
    {
        r_SpreadEntryPriceHasChanged = 0;

        Input_IsActiveAndWaitingForSpreadTriggerPrice.SetYesNo(false);
        r_CurrentState = STATE_DISABLED_INVALID_ENTRY_PRICE;
        r_CurrentOnScreenMessage = "Disabled: DisableIfImmediateFillOfSpreadAfterSpreadPriceSet";

        TradingSpreadOrderEntry_DrawStatusText(sc);
        return;
    }

    r_SpreadEntryPriceHasChanged = 0;
}
else
{
    return;
}

if (!Input_IsActiveAndWaitingForSpreadTriggerPrice.GetYesNo())
    return; //this will not be reached but is here for safety.

if (r_CurrentState != STATE_MONITORING_FOR_SPREAD_PRICE_TRIGGER
    && r_CurrentState != STATE_MONITORING_FOR_TARGET_AND_STOP)
    return;

//From this point on *must* get to end of function for safety so state will be set to disabled or monitoring for target or
stop.

r_CurrentOnScreenMessage += sc.FormatDateTime(sc.CurrentSystemDateTime);
r_CurrentOnScreenMessage += ". ";

bool Successful = false;
for (int SpreadOrderDataIndex = 0; SpreadOrderDataIndex < NUM_SPREAD_ORDERS; ++SpreadOrderDataIndex)
{
    if (SpreadQuantities[SpreadOrderDataIndex] == 0.0)
        continue;

    if (SpreadSymbols[SpreadOrderDataIndex].IsEmpty())
        continue;

    if (SpreadOrderDataIndex > 0)
        r_CurrentOnScreenMessage += ", ";

    // At this point it is safe to submit the market order

```

```

s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = SpreadQuantities[SpreadOrderDataIndex];
NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
NewOrder.TimeInForce = SCT_TIF_DAY;

NewOrder.Symbol = SpreadSymbols[SpreadOrderDataIndex];
NewOrder.TradeAccount = sc.SelectedTradeAccount;
NewOrder.TextTag = "Spread order entry leg";

int Result = 0;
if (r_CurrentState == STATE_MONITORING_FOR_TARGET_AND_STOP && UseTargetOrStop)
{
    if (SpreadSides[SpreadOrderDataIndex] == 0)//Buy
        Result = static_cast<int>(sc.SellOrder(NewOrder));
    else if (SpreadSides[SpreadOrderDataIndex] == 1)//Sell
        Result = static_cast<int>(sc.BuyOrder(NewOrder));
}
else
{
    if (SpreadSides[SpreadOrderDataIndex] == 0)//Buy
        Result = static_cast<int>(sc.BuyOrder(NewOrder));
    else if (SpreadSides[SpreadOrderDataIndex] == 1)//Sell
        Result = static_cast<int>(sc.SellOrder(NewOrder));
}

if (Result < 0)
{
    r_CurrentOnScreenMessage = "Submission error: ";
    r_CurrentOnScreenMessage += sc.GetTradingErrorTextMessage(Result);
}
else
{
    Successful = true;
    r_CurrentOnScreenMessage += NewOrder.Symbol;
    r_CurrentOnScreenMessage += "**";
    if (r_CurrentState == STATE_MONITORING_FOR_TARGET_AND_STOP && UseTargetOrStop)
    {
        if (SpreadSides[SpreadOrderDataIndex] == 0)
            r_CurrentOnScreenMessage += "-";
    }
    else
    {
        if (SpreadSides[SpreadOrderDataIndex] == 1)
            r_CurrentOnScreenMessage += "-";
    }

    r_CurrentOnScreenMessage.AppendFormat("%0.2f", NewOrder.OrderQuantity);

    SCString MessageText("Market order submitted for spread leg ");
    MessageText += NewOrder.Symbol;
    sc.AddMessageToTradeServiceLog(MessageText, false);
}
}

//Disable or monitor for target or stop

if (!UseTargetOrStop)
{
    Input_IsActiveAndWaitingForSpreadTriggerPrice.SetYesNo(false);//Disable
    r_CurrentOnScreenMessage += ". Not Monitoring";
    r_CurrentState = STATE_DISABLED_AFTER_ORDERS_SENT;
}

```

```

    Input_IsActiveAndWaitingForSpreadTriggerPrice.SetYesNo(false);//Disable
}
else
{
    if (r_CurrentState == STATE_MONITORING_FOR_SPREAD_PRICE_TRIGGER)
    {
        r_CurrentOnScreenMessage += ". Monitoring Target/Stop";
        r_CurrentState = STATE_MONITORING_FOR_TARGET_AND_STOP;
    }
    else
    {
        r_CurrentOnScreenMessage += ". Not Monitoring";
        r_CurrentState = STATE_DISABLED_AFTER_ORDERS_SENT;
        Input_IsActiveAndWaitingForSpreadTriggerPrice.SetYesNo(false);//Disable
    }
}

TradingSpreadOrderEntry_DrawStatusText(sc);
}

/*=====*/
void TradingSpreadOrderEntry_DrawStatusText(SCStudyInterfaceRef sc )
{
    SCSubgraphRef Subgraph_OnChartTextDisplay = sc.Subgraph[1];

    int &r_DrawingNumber = sc.GetPersistentInt(5);

    SCString& r_CurrentOnScreenMessage = sc.GetPersistentSCString(1);

    s_UseTool Tool;

    Tool.Clear();
    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_TEXT;
    Tool.BeginDateTime = 10;
    Tool.Region = sc.GraphRegion;
    Tool.BeginValue = 90;
    Tool.UseRelativeVerticalValues = true;
    Tool.Color = Subgraph_OnChartTextDisplay.PrimaryColor;
    Tool.FontBold = true;

    Tool.Text = r_CurrentOnScreenMessage;
    Tool.FontSize = Subgraph_OnChartTextDisplay.LineWidth;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;
    Tool.ReverseTextColor = false;

    if (r_DrawingNumber != 0)
        Tool.LineNumber = r_DrawingNumber;

    if(sc.UseTool(Tool))
        r_DrawingNumber = Tool.LineNumber;
}

/*=====*/
void TradingSpreadOrderEntry_UpdateTargetStopSubgraphs(SCStudyInterfaceRef sc, int BeginIndex, int EndIndex)
{
    SCSubgraphRef Subgraph_TargetPriceLine = sc.Subgraph[3];
    SCSubgraphRef Subgraph_StopPriceLine = sc.Subgraph[4];

    SCInputRef Input_UseTargetAfterFillOfSpreadEntry = sc.Input[17];

```

```
SCInputRef Input_TargetSpreadPrice = sc.Input[18];

SCInputRef Input_UseStopAfterFillOfSpreadEntry = sc.Input[19];
SCInputRef Input_StopSpreadPrice = sc.Input[20];

for (int BarIndex = BeginIndex; BarIndex < EndIndex; BarIndex++)
{
    if (Input_UseTargetAfterFillOfSpreadEntry.GetYesNo())
        Subgraph_TargetPriceLine[BarIndex] = Input_TargetSpreadPrice.GetFloat();

    if (Input_UseStopAfterFillOfSpreadEntry.GetYesNo())
        Subgraph_StopPriceLine[BarIndex] = Input_StopSpreadPrice.GetFloat();
}
}
```