

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

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

```
SCSFExport scsf_BidAskQuantityTriggeredStopOrder(SCStudyInterfaceRef sc)
```

```
{
    SCSubgraphRef Subgraph_OrderLine = sc.Subgraph[0];
    SCInputRef Input_NumberOfBarsBack = sc.Input[3];
    SCInputRef Input_BidAskQuantityThreshold = sc.Input[4];
    SCInputRef Input_DebuggingOutput = sc.Input[5];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Trading: Bid/Ask Quantity Triggered Stop Order";
        sc.AutoLoop = 0; //Manual looping
        sc.GraphRegion = 0;
        sc.ScaleRangeType= SCALE_SAMEASREGION;

        Subgraph_OrderLine.Name = "Stop Order";
        Subgraph_OrderLine.PrimaryColor = COLOR_GREEN;
        Subgraph_OrderLine.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_LEFT_EDGE |
LL_VALUE_ALIGN_BELOW | LL_DISPLAY_NAME | LL_NAME_ALIGN_LEFT_EDGE | LL_NAME_ALIGN_ABOVE;
        Subgraph_OrderLine.DrawZeros = false;

        Input_NumberOfBarsBack.Name = "Number of Bars Back to Display";
        Input_NumberOfBarsBack.SetInt(20);

        Input_BidAskQuantityThreshold.Name = "Bid Ask Quantity Threshold";
        Input_BidAskQuantityThreshold.SetInt(100);

        Input_DebuggingOutput.Name = "Debugging Output";
        Input_DebuggingOutput.SetYesNo(false);

        sc.ReceivePointerEvents = ACS_RECEIVE_POINTER_EVENTS_ALWAYS;

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

        //use a high-value to effectively not put any Position Quantity restrictions
        sc.MaximumPositionAllowed = 100000000;
        return;
    }

    int& r_SetSellOrderPriceMenuID = sc.GetPersistentInt(1);
    int& r_SetBuyOrderPriceMenuID = sc.GetPersistentInt(2);
    int& r_CancelOrderMenuID = sc.GetPersistentInt(3);
    int& r_OrderType = sc.GetPersistentInt(4);
    int& r_LastSubgraphUpdateIndex = sc.GetPersistentInt(5);
    double& r_OrderPrice = sc.GetPersistentDouble(6);
    int& r_OrderModifyMode = sc.GetPersistentInt(7);
    int& r_LastDebugMessageType = sc.GetPersistentInt(8);
    int& r_OrderEvaluationCount = sc.GetPersistentInt(9);
    int& r_StopOrderActive = sc.GetPersistentInt(10);

    __int64& r_LastTimeAndSalesRecordSequence = sc.GetPersistentInt64(11);

    if (sc.LastCallToFunction)
    {
```

```

// Be sure to remove the menu commands when study is removed
sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_SetBuyOrderPriceMenuID);
sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_SetSellOrderPriceMenuID);
sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID);
return;
}

if (sc.UpdateStartIndex == 0)
{
    if (r_SetBuyOrderPriceMenuID <= 0)
        r_SetBuyOrderPriceMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Set Buy Stop Price");

    if (r_SetSellOrderPriceMenuID <= 0)
        r_SetSellOrderPriceMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Set Sell Stop Price");

    r_LastSubgraphUpdateIndex = 0;

    r_LastDebugMessageType = 0;

    r_LastTimeAndSalesRecordSequence = 0;
}

if (sc.MenuEventID != 0)
{
    if (!sc.ChartTradeModeEnabled)
    {
        sc.AddMessageToLog("Chart Trade Mode is not active. No action performed.", 1);
        return;
    }
}

// handle set trigger price menu clicks
if (sc.MenuEventID != 0 && sc.MenuEventID == r_SetBuyOrderPriceMenuID)
{
    // Get price that the user selected. This is already rounded to the nearest price tick
    r_OrderPrice = sc.ChartTradingOrderPrice;
    r_OrderType = 1; // Buy

    if (r_CancelOrderMenuID <= 0)
        r_CancelOrderMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Cancel Stop Order");

    Subgraph_OrderLine.Name.Format("Buy Stop (%d)", Input_BidAskQuantityThreshold.GetInt());
}
else if (sc.MenuEventID != 0 && sc.MenuEventID == r_SetSellOrderPriceMenuID)
{
    // Get price that the user selected. This is already rounded to the nearest price tick
    r_OrderPrice = sc.ChartTradingOrderPrice;
    r_OrderType = -1; // Sell

    if (r_CancelOrderMenuID <= 0)
        r_CancelOrderMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Cancel Stop Order");

    Subgraph_OrderLine.Name.Format("Sell Stop (%d)", Input_BidAskQuantityThreshold.GetInt());
}

// handle cancel order menu clicks
if (sc.MenuEventID != 0 && sc.MenuEventID == r_CancelOrderMenuID)
{
    r_OrderType = 0;
    r_OrderPrice = 0.0;
}

```

```

r_StopOrderActive = 0;

sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID) ;
r_CancelOrderMenuID = 0;
r_LastTimeAndSalesRecordSequence = 0;
}

//Handle order modifications
if (r_OrderType != 0)
{
    if (sc.PointerEventType == SC_POINTER_BUTTON_DOWN
        && r_OrderPrice >= sc.ChartTradingOrderPrice - sc.TickSize
        && r_OrderPrice <= sc.ChartTradingOrderPrice + sc.TickSize)
        r_OrderModifyMode = 1;
    else if (sc.PointerEventType == SC_POINTER_BUTTON_UP)
        r_OrderModifyMode = 0;

    if (r_OrderModifyMode == 1 && sc.PointerEventType == SC_POINTER_MOVE)
    {
        r_OrderPrice = sc.ChartTradingOrderPrice;
    }
}

//Update the subgraph
int StartingOutputIndex = sc.ArraySize - Input_NumberOfBarsBack.GetInt();

for (int BarIndex = r_LastSubgraphUpdateIndex; BarIndex < StartingOutputIndex; BarIndex++)
{
    Subgraph_OrderLine[BarIndex] = 0.0f;
}

r_LastSubgraphUpdateIndex = StartingOutputIndex;

for (int BarIndex = StartingOutputIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    Subgraph_OrderLine[BarIndex] = static_cast<float>(r_OrderPrice);
}

//Evaluate for order triggering

//For safety we must never do any order management while historical data is being downloaded.
if (sc.ChartIsDownloadingHistoricalData(sc.ChartNumber))
{
    if (Input_DebuggingOutput.GetYesNo() && r_LastDebugMessageType != 1)
    {
        sc.AddMessageToLog("Chart is downloading historical data. Order evaluation not performed.", 0);
        r_LastDebugMessageType = 1;
    }
    return;
}

if (r_OrderType == 0)
{
    if (Input_DebuggingOutput.GetYesNo() && r_LastDebugMessageType != 2)
    {
        sc.AddMessageToLog("Order price is not set. Order evaluation not performed.", 0);
        r_LastDebugMessageType = 2;
    }

    return;
}

```

```

sc.SendOrdersToTradeService = !sc.GlobalTradeSimulationIsOn;

bool PerformReset = false;

//Iterate through new time and sales data
c_SCTimeAndSalesArray TimeSalesArray;
sc.GetTimeAndSales(TimeSalesArray);

if (TimeSalesArray.Size() == 0)
    return;//do nothing.

if (r_LastTimeAndSalesRecordSequence == 0)
{
    r_LastTimeAndSalesRecordSequence = TimeSalesArray[TimeSalesArray.Size()-1].Sequence;
    return;
}

for (int TSIndex = 0; TSIndex < TimeSalesArray.Size(); ++TSIndex)
{
    s_TimeAndSales Record = TimeSalesArray[TSIndex];
    Record *= sc.RealTimePriceMultiplier;

    //Do not process already processed sequence numbers.
    if (Record.Sequence <= r_LastTimeAndSalesRecordSequence)
        continue;

    r_LastTimeAndSalesRecordSequence = Record.Sequence;

    if (Record.Type != SC_TS_ASK && Record.Type != SC_TS_BID)
        continue;

    //Check for triggering. Order price needs to equal bid or ask to be triggered.
    if (!r_StopOrderActive)
    {
        if (r_OrderType == -1 )
        {
            if (sc.FormattedEvaluateUsingDoubles( r_OrderPrice, sc.BaseGraphValueFormat, EQUAL_OPERATOR,
sc.Bid , sc.BaseGraphValueFormat))
                r_StopOrderActive = 1;
        }
        else if (r_OrderType == 1 )
        {
            if (sc.FormattedEvaluateUsingDoubles( r_OrderPrice, sc.BaseGraphValueFormat, EQUAL_OPERATOR,
sc.Ask , sc.BaseGraphValueFormat))
                r_StopOrderActive = 1;
        }
    }
}

// Check for sell stop order trigger conditions. Best Bid Quantity if it is a sell stop
if (r_StopOrderActive != 0 && r_OrderType == -1 && sc.Bid != 0.0)
{
    float CurrentPriceToOrderPriceDifference = static_cast<float>(r_OrderPrice) - sc.Bid;

    if ((sc.FormattedEvaluate( CurrentPriceToOrderPriceDifference, sc.BaseGraphValueFormat, EQUAL_OPERATOR,
0.0 , sc.BaseGraphValueFormat)
        && sc.BidSize <= Input_BidAskQuantityThreshold.GetInt())

        || (sc.FormattedEvaluate( CurrentPriceToOrderPriceDifference, sc.BaseGraphValueFormat,
GREATER_OPERATOR, 0.0 , sc.BaseGraphValueFormat)
            && sc.FormattedEvaluate( CurrentPriceToOrderPriceDifference, sc.BaseGraphValueFormat,
LESS_EQUAL_OPERATOR, static_cast<float>(8) * sc.TickSize , sc.BaseGraphValueFormat))

```

```

    )
    {

        if (Input_DebuggingOutput.GetYesNo())
            sc.AddMessageToLog("Sell stop is triggered. Sending sell Market order.", 0);

        s_SCNewOrder Order;
        //Need to use a market order type since the price triggering is determined within this function and to avoid
        possible rejection of a stop order by the external trading service based upon the price.
        Order.OrderType = SCT_ORDERTYPE_MARKET;
        Order.OrderQuantity = sc.TradeWindowOrderQuantity;

        int Result = static_cast<int>(sc.SellOrder(Order));
        PerformReset = true;

        if (Result <= 0)
        {
            SCString ErrorText;
            ErrorText = "Sell order error. ";
            ErrorText += sc.GetTradingErrorTextMessage(Result);
            sc.AddMessageToLog(ErrorText, 1);
        }
    }
    else
    {
        if (Input_DebuggingOutput.GetYesNo() && r_OrderEvaluationCount % 10 == 0)
        {
            SCString MessageText;
            MessageText.Format( "Sell stop order not triggered. Bid=%f, OrderPrice=%f, BidQuantity=%d,
            BidQuantityThreshold=%d.", sc.Bid, r_OrderPrice, sc.BidSize , Input_BidAskQuantityThreshold.GetInt());
            sc.AddMessageToLog(MessageText, 0);
        }
    }
}

// Check for buy stop order trigger conditions. Best Ask Quantity if it is a buy stop
if (r_StopOrderActive != 0 && r_OrderType == 1 && sc.Ask != 0.0)
{
    float CurrentPriceToOrderPriceDifference = sc.Ask - static_cast<float>(r_OrderPrice);

    if ((sc.FormattedEvaluate( CurrentPriceToOrderPriceDifference, sc.BaseGraphValueFormat, EQUAL_OPERATOR,
    0.0 , sc.BaseGraphValueFormat)
        && sc.AskSize <= Input_BidAskQuantityThreshold.GetInt())

        || (sc.FormattedEvaluate( CurrentPriceToOrderPriceDifference, sc.BaseGraphValueFormat,
        GREATER_OPERATOR, 0.0 , sc.BaseGraphValueFormat)
            && sc.FormattedEvaluate( CurrentPriceToOrderPriceDifference, sc.BaseGraphValueFormat,
            LESS_EQUAL_OPERATOR, static_cast<float>(8) * sc.TickSize , sc.BaseGraphValueFormat))

    )
    {

        if (Input_DebuggingOutput.GetYesNo())
            sc.AddMessageToLog("Buy stop is triggered. Sending buy Market order.", 0);

        s_SCNewOrder Order;
        Order.OrderType = SCT_ORDERTYPE_MARKET;
        Order.OrderQuantity = sc.TradeWindowOrderQuantity;

        int Result = static_cast<int>(sc.BuyOrder(Order));
        PerformReset = true;

        if (Result <= 0)
        {
            SCString ErrorText;

```

```

        ErrorText = "Buy order error. ";
        ErrorText += sc.GetTradingErrorTextMessage(Result);
        sc.AddMessageToLog(ErrorText, 1);
    }
}
else
{
    if (Input_DebuggingOutput.GetYesNo() && r_OrderEvaluationCount % 10 == 0)
    {
        SCString MessageText;
        MessageText.Format( "Buy stop order not triggered. Ask=%f, OrderPrice=%f, AskQuantity=%d,
AskQuantityThreshold=%d.", sc.Ask, r_OrderPrice, sc.AskSize , Input_BidAskQuantityThreshold.GetInt());
        sc.AddMessageToLog(MessageText, 0);
    }
}
}

++r_OrderEvaluationCount;

if (PerformReset)
{
    sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID);

    r_CancelOrderMenuID = 0;

    r_OrderPrice = 0.0;
    r_OrderType = 0;
    r_OrderEvaluationCount = 0;
    r_StopOrderActive = 0;

    for (int ArrayIndex = r_LastSubgraphUpdateIndex; ArrayIndex < sc.ArraySize; ArrayIndex++)
        Subgraph_OrderLine[ArrayIndex] = 0;
}
}

/*=====*/
SCSFExport scsf_MarketIfTouchedOrder(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_OrderLine = sc.Subgraph[0];
    SCInputRef Input_NumberOfBarsBack = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Trading: Market if Touched Order";
        sc.AutoLoop = 0; //Manual looping
        sc.GraphRegion = 0;
        sc.ScaleRangeType= SCALE_SAMEASREGION;

        Input_NumberOfBarsBack.Name = "Number of Bars Back to Display";
        Input_NumberOfBarsBack.SetInt(20);

        Subgraph_OrderLine.Name = "MIT Order";
        Subgraph_OrderLine.PrimaryColor = COLOR_GREEN;
        Subgraph_OrderLine.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_LEFT_EDGE |
LL_VALUE_ALIGN_BELOW | LL_DISPLAY_NAME | LL_NAME_ALIGN_LEFT_EDGE | LL_NAME_ALIGN_ABOVE;
        Subgraph_OrderLine.DrawZeros = false;

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

```

```

sc.SupportAttachedOrdersForTrading = 0;
sc.UseGUIAttachedOrderSetting = 1;

//use a high-value to effectively not put any Position Quantity restrictions
sc.MaximumPositionAllowed = 100000000;

sc.ReceivePointerEvents = ACS_RECEIVE_POINTER_EVENTS_ALWAYS;

return;
}

int& r_SetSellOrderPriceMenuID = sc.GetPersistentInt(1);
int& r_SetBuyOrderPriceMenuID = sc.GetPersistentInt(2);
int& r_CancelOrderMenuID = sc.GetPersistentInt(3);
int& r_OrderType = sc.GetPersistentInt(4);
int& r_LastSubgraphUpdateIndex = sc.GetPersistentInt(5);
double& r_OrderPrice = sc.GetPersistentDouble(6);
int& r_OrderModifyMode = sc.GetPersistentInt(7);

if (sc.LastCallToFunction)
{
    // Be sure to remove the menu commands when study is removed
    sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_SetBuyOrderPriceMenuID);
    sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_SetSellOrderPriceMenuID);
    sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID);
    return;
}

if (sc.UpdateStartIndex == 0)
{
    if (r_SetBuyOrderPriceMenuID <= 0)
        r_SetBuyOrderPriceMenuID = sc.AddACSChartShortcutMenuItem(sc.ChartNumber, "Set Buy MIT Price");

    if (r_SetSellOrderPriceMenuID <= 0)
        r_SetSellOrderPriceMenuID = sc.AddACSChartShortcutMenuItem(sc.ChartNumber, "Set Sell MIT Price");

    r_LastSubgraphUpdateIndex = 0;
}

if (sc.MenuEventID != 0)
{
    if (!sc.ChartTradeModeEnabled)
    {
        sc.AddMessageToLog("Chart Trade Mode is not active. No action performed.", 1);
        return;
    }
}

// handle set order price menu clicks
if (sc.MenuEventID != 0 && sc.MenuEventID == r_SetBuyOrderPriceMenuID)
{
    // Get price that the user selected. This is already rounded to the nearest price tick
    r_OrderPrice = sc.ChartTradingOrderPrice;
    r_OrderType = 1; // Buy

    if (r_CancelOrderMenuID <= 0)
        r_CancelOrderMenuID = sc.AddACSChartShortcutMenuItem(sc.ChartNumber, "Cancel MIT Order");

    Subgraph_OrderLine.Name = "Buy MIT Order";
}
else if (sc.MenuEventID != 0 && sc.MenuEventID == r_SetSellOrderPriceMenuID)

```

```

{
    // Get price that the user selected. This is already rounded to the nearest price tick
    r_OrderPrice = sc.ChartTradingOrderPrice;
    r_OrderType = -1; // Sell

    if (r_CancelOrderMenuID <= 0)
        r_CancelOrderMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Cancel MIT Order");

    Subgraph_OrderLine.Name = "Sell MIT Order";
}

// handle cancel order menu clicks
if (sc.MenuEventID != 0 && sc.MenuEventID == r_CancelOrderMenuID)
{
    r_OrderType = 0;
    r_OrderPrice = 0.0;

    sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID) ;
    r_CancelOrderMenuID = 0;
}

//Handle order modifications
if (r_OrderType != 0)
{
    if(sc.PointerEventType == SC_POINTER_BUTTON_DOWN
        && r_OrderPrice >= sc.ChartTradingOrderPrice - sc.TickSize
        && r_OrderPrice <= sc.ChartTradingOrderPrice + sc.TickSize)
        r_OrderModifyMode = 1;
    else if(sc.PointerEventType == SC_POINTER_BUTTON_UP)
        r_OrderModifyMode = 0;

    if (r_OrderModifyMode == 1 && sc.PointerEventType == SC_POINTER_MOVE)
    {
        r_OrderPrice = sc.ChartTradingOrderPrice;
    }
}

//Update the subgraph
int StartingOutputIndex = sc.ArraySize - Input_NumberOfBarsBack.GetInt();

for (int BarIndex = r_LastSubgraphUpdateIndex; BarIndex < StartingOutputIndex; BarIndex++)
{
    Subgraph_OrderLine[BarIndex] = 0.0f;
}

r_LastSubgraphUpdateIndex = StartingOutputIndex;

for (int BarIndex = StartingOutputIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    Subgraph_OrderLine[BarIndex] = static_cast<float>(r_OrderPrice);
}

//Evaluate for order triggering

//For safety we must never do any order management while historical data is being downloaded.
if (sc.ChartIsDownloadingHistoricalData(sc.ChartNumber))
    return;

// If the order is zero there is nothing to do
if (r_OrderPrice == 0.0 || sc.LastTradePrice == 0.0)

```



```

return;

sc.SendOrdersToTradeService = !sc.GlobalTradeSimulationIsOn;

bool PerformReset = false;

// check for sell order trigger conditions
if (r_OrderType == -1
    && sc.FormattedEvaluate( sc.LastTradePrice, sc.BaseGraphValueFormat, GREATER_EQUAL_OPERATOR,
static_cast<float>(r_OrderPrice), sc.BaseGraphValueFormat))
{

    s_SCNewOrder Order;
    Order.OrderType = SCT_ORDERTYPE_MARKET;
    Order.OrderQuantity = sc.TradeWindowOrderQuantity;

    int Result = static_cast<int>(sc.SellOrder(Order));
    if (Result > 0)
        PerformReset = true;

}

// check for buy order trigger conditions
if (r_OrderType == 1
    && sc.FormattedEvaluate(sc.LastTradePrice, sc.BaseGraphValueFormat, LESS_EQUAL_OPERATOR,
static_cast<float>(r_OrderPrice), sc.BaseGraphValueFormat))
{

    s_SCNewOrder Order;
    Order.OrderType = SCT_ORDERTYPE_MARKET;
    Order.OrderQuantity = sc.TradeWindowOrderQuantity;

    int Result = static_cast<int>(sc.BuyOrder(Order));
    if (Result > 0)
        PerformReset = true;

}

if (PerformReset)
{
    sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID);

    r_CancelOrderMenuID = 0;
    r_OrderPrice = 0.0;
    r_OrderType = 0;

    for (int ArrayIndex = r_LastSubgraphUpdateIndex; ArrayIndex < sc.ArraySize; ArrayIndex++)
        Subgraph_OrderLine[ArrayIndex] = 0;
}

}

/*=====*/
SCSFExport scsf_TradingTriggeredStopOrderEntry(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_TriggerLine = sc.Subgraph[0];
    SCSubgraphRef Subgraph_OrderLine = sc.Subgraph[1];
    SCInputRef Input_NumberOfBarsBack = sc.Input[0];
    SCInputRef Input_AutoSetStopPrice = sc.Input[1];
    SCInputRef Input_StopPriceOffsetInTicks = sc.Input[2];

    if (sc.SetDefaults)

```

```

{
    sc.GraphName = "Trading: Triggered Stop Order";
    sc.AutoLoop = 0;//Manual looping
    sc.GraphRegion = 0;
    sc.ScaleRangeType= SCALE_SAMEASREGION;

    Input_NumberOfBarsBack.Name = "Number of Bars Back to Display";
    Input_NumberOfBarsBack.SetInt(20);

    //AutoSetStopPrice.Name = "Auto Set Stop Price";
    //AutoSetStopPrice.SetYesNo(true);

    Input_StopPriceOffsetInTicks.Name = "Stop Price Offset from Trigger Price in Ticks";
    Input_StopPriceOffsetInTicks.SetInt(4);

    Subgraph_OrderLine.Name = "Order Line";
    Subgraph_OrderLine.PrimaryColor = COLOR_GREEN;
    Subgraph_OrderLine.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_LEFT_EDGE |
LL_VALUE_ALIGN_BELOW | LL_DISPLAY_NAME | LL_NAME_ALIGN_LEFT_EDGE | LL_NAME_ALIGN_ABOVE;
    Subgraph_OrderLine.DrawZeros = false;

    Subgraph_TriggerLine.Name = "Stop Order Trigger";
    Subgraph_TriggerLine.PrimaryColor = COLOR_CYAN;
    Subgraph_TriggerLine.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_LEFT_EDGE |
LL_VALUE_ALIGN_BELOW | LL_DISPLAY_NAME | LL_NAME_ALIGN_LEFT_EDGE | LL_NAME_ALIGN_ABOVE;
    Subgraph_TriggerLine.DrawZeros = false;

    sc.ReceivePointerEvents = false;

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

    //use a high-value to effectively not put any Position Quantity restrictions
    sc.MaximumPositionAllowed = 100000000;

    return;
}

int& r_SetTriggerPriceMenuID = sc.GetPersistentInt(1);
int& r_CancelOrderMenuID = sc.GetPersistentInt(2);
int& r_OrderType = sc.GetPersistentInt(3);//1 = buy stop, -1 = sell stop
int& r_LastSubgraphUpdateIndex = sc.GetPersistentInt(4);
double& r_TriggerPrice = sc.GetPersistentDouble(5);
double& r_OrderPrice = sc.GetPersistentDouble(6);

if (sc.LastCallToFunction)
{
    // Be sure to remove the menu commands when study is removed
    sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_SetTriggerPriceMenuID);
    sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID);
    return;
}

if (sc.UpdateStartIndex == 0)//Perform initialization
{
    if (r_SetTriggerPriceMenuID <= 0)

```

```

    {
        r_SetTriggerPriceMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Set Stop Trigger Price");
    }

    r_LastSubgraphUpdateIndex = 0;
}

//Handle pointer events from the user
if (sc.MenuEventID != 0)
{
    if (!sc.ChartTradeModeEnabled)
    {
        sc.AddMessageToLog("Chart Trade Mode is not active. No action performed.", 1);
        return;
    }
}

// handle set trigger price menu clicks
if (sc.MenuEventID != 0 && sc.MenuEventID == r_SetTriggerPriceMenuID)
{
    const double CurrentPrice = sc.GetLastPriceForTrading();

    // Get price that the user selected. This is already rounded to the nearest price tick
    r_TriggerPrice = sc.ChartTradingOrderPrice;

    if (sc.ChartTradingOrderPrice >= CurrentPrice) // Buy
    {
        r_OrderPrice = sc.ChartTradingOrderPrice + Input_StopPriceOffsetInTicks.GetInt() * sc.TickSize;
        r_OrderType = 1;
    }
    else // Sell
    {
        r_OrderPrice = sc.ChartTradingOrderPrice - Input_StopPriceOffsetInTicks.GetInt() * sc.TickSize;
        r_OrderType = -1;
    }

    if (r_CancelOrderMenuID == 0)
        r_CancelOrderMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Cancel Stop Order");
}

// handle cancel order menu clicks
if (sc.MenuEventID != 0 && sc.MenuEventID == r_CancelOrderMenuID)
{
    r_OrderType = 0;
    r_OrderPrice = 0.0;
    r_TriggerPrice = 0.0;

    sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID);
    r_CancelOrderMenuID = 0;
}

//Update the horizontal subgraph line
int StartingOutputIndex = sc.ArraySize - Input_NumberOfBarsBack.GetInt();

for (int BarIndex = r_LastSubgraphUpdateIndex; BarIndex < StartingOutputIndex; BarIndex++)
{
    Subgraph_TriggerLine[BarIndex] = 0.0f;
}

```

```

    Subgraph_OrderLine[BarIndex] = 0.0f;
}

r_LastSubgraphUpdateIndex = StartingOutputIndex;

for (int BarIndex = StartingOutputIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    Subgraph_TriggerLine[BarIndex] = static_cast<float>(r_TriggerPrice);
    Subgraph_OrderLine[BarIndex] = static_cast<float>(r_OrderPrice);
}

if (r_OrderType == -1)
    Subgraph_OrderLine.Name = "Sell Stop Order";
else if (r_OrderType == 1)
    Subgraph_OrderLine.Name = "Buy Stop Order";

//For safety must never do any order management while historical data is being downloaded.
if (sc.ChartIsDownloadingHistoricalData(sc.ChartNumber))
    return;

// If the order price is zero there is nothing to do
if (r_OrderPrice == 0.0 || r_OrderType == 0)
    return;

sc.SendOrdersToTradeService = !sc.GlobalTradeSimulationIsOn;

bool PerformReset = false;

// check for sell order trigger conditions
if (r_OrderType == -1
    && sc.FormattedEvaluate(sc.LastTradePrice, sc.BaseGraphValueFormat, LESS_EQUAL_OPERATOR,
static_cast<float>(r_TriggerPrice), sc.BaseGraphValueFormat))
{
    s_SCNewOrder Order;
    Order.Price1 = r_OrderPrice;
    Order.OrderType = SCT_ORDERTYPE_STOP;
    Order.OrderQuantity = sc.TradeWindowOrderQuantity;

    int Result = static_cast<int>(sc.SellOrder(Order));
    if (Result > 0)
        PerformReset = true;
}

// check for buy order trigger conditions
if (r_OrderType == 1
    && sc.FormattedEvaluate( sc.LastTradePrice, sc.BaseGraphValueFormat, GREATER_EQUAL_OPERATOR,
static_cast<float>(r_TriggerPrice), sc.BaseGraphValueFormat))
{
    s_SCNewOrder Order;
    Order.Price1 = r_OrderPrice;
    Order.OrderType = SCT_ORDERTYPE_STOP;
    Order.OrderQuantity = sc.TradeWindowOrderQuantity;

    int Result = static_cast<int>(sc.BuyOrder(Order));
    if (Result > 0)
        PerformReset = true;
}

if (PerformReset)
{
    if (r_CancelOrderMenuID != 0)

```

```

sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID);

r_CancelOrderMenuID = 0;

r_OrderPrice = 0.0;
r_TriggerPrice = 0.0;
r_OrderType = 0;

for (int ArrayIndex = r_LastSubgraphUpdateIndex; ArrayIndex < sc.ArraySize; ArrayIndex++)
{
    Subgraph_TriggerLine[ArrayIndex] = 0;
    Subgraph_OrderLine[ArrayIndex] = 0;
}
}
}

/*=====*/
/******/
/* XYZ has a market price of $16.45. You decide to buy 100 shares, but you don't want to pay more than $16.35 and you
don't want to enter the market until the price drops to $16.40. Create a BUY order, and select LIT in the Type field to
specify a limit if touched order. In the Lmt Price field, enter a limit price of $16.35, and in the Trigger Price field, enter the
trigger price of $16.40. Transmit the order, which will be held in the system until the trigger price is touched, and will then
be submitted as a limit order. It will only execute at $16.35 or better. */
/******/
SCSFExport scsf_LimitIfTouchedOrder(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_OrderLine = sc.Subgraph[0];
    SCInputRef Input_NumberOfBarsBack = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Trading: Limit if Touched Order";
        sc.AutoLoop = 0; //Manual looping
        sc.GraphRegion = 0;
        sc.ScaleRangeType = SCALE_SAMEASREGION;

        Input_NumberOfBarsBack.Name = "Number of Bars Back to Display";
        Input_NumberOfBarsBack.SetInt(20);

        Subgraph_OrderLine.Name = "LIT Order";
        Subgraph_OrderLine.PrimaryColor = COLOR_GREEN;
        Subgraph_OrderLine.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_LEFT_EDGE |
LL_VALUE_ALIGN_BELOW | LL_DISPLAY_NAME | LL_NAME_ALIGN_LEFT_EDGE | LL_NAME_ALIGN_ABOVE;
        Subgraph_OrderLine.DrawZeros = false;

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

        //use a high-value to effectively not put any Position Quantity restrictions
        sc.MaximumPositionAllowed = 100000000;

        sc.ReceivePointerEvents = ACS_RECEIVE_POINTER_EVENTS_ALWAYS;

        return;
    }
}

```

}

/\*=====\*/