

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

```
/*=====*/
void TradingTriggeredLimitOrderEntry_UpdateTriggerAndOrderPriceTextDisplay(SCStudyInterfaceRef sc)
{
    SCSubgraphRef TextDisplay = sc.Subgraph[2];
    SCInputRef TextHorizontalPosition = sc.Input[0];
    SCInputRef TextVerticalPosition = sc.Input[1];

    //Make sure the keys match with the primary function
    int& r_LineNumber = sc.GetPersistentInt(9);
    double& r_TriggerPrice = sc.GetPersistentDouble(9);

    s_UseTool Tool;

    Tool.Clear();
    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_TEXT;

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

    Tool.BeginDateTime = TextHorizontalPosition.GetInt();

    Tool.Region = sc.GraphRegion;
    Tool.BeginValue = static_cast<float>(TextVerticalPosition.GetInt());
    Tool.UseRelativeVerticalValues = true;
    Tool.Color = TextDisplay.PrimaryColor;
    Tool.FontBackColor = TextDisplay.SecondaryColor;
    Tool.FontBold = true;

    SCString Text;

    Text.Format("Limit Trigger: ");
    if (r_TriggerPrice != 0.0) {
        Text += sc.FormatGraphValue(r_TriggerPrice, sc.GetValueFormat());
    }
    else
    {
        Text += "Not Set";
    }

    Tool.Text = Text;
    Tool.FontSize = TextDisplay.LineWidth;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;
    Tool.ReverseTextColor = false;

    sc.UseTool(Tool);

    r_LineNumber = Tool.LineNumber;
}
/*=====*/
SCSFExport scsf_TradingTriggeredLimitOrderEntry(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_TriggerLine = sc.Subgraph[0];
    SCSubgraphRef Subgraph_OrderLine = sc.Subgraph[1];
    SCSubgraphRef Subgraph_TextDisplay = sc.Subgraph[2];

    SCInputRef Input_TextHorizontalPosition = sc.Input[0];
    SCInputRef Input_TextVerticalPosition = sc.Input[1];
    SCInputRef Input_NumberOfBarsBack = sc.Input[3];
    SCInputRef Input_AutoSetLimitPrice = sc.Input[4];
    SCInputRef Input_LimitPriceOffsetInTicks = sc.Input[5];
```

```

if (sc.SetDefaults)
{
    sc.GraphName = "Trading: Triggered Limit Order Entry";
    sc.StudyDescription = "This study provides an example of how to implement custom order management based on
price action within Sierra Chart. The study uses the menu API to provide user interaction.";
    sc.AutoLoop = 0; //Manual looping
    sc.GraphRegion = 0;
    sc.ScaleRangeType = SCALE_SAMEASREGION;

    Subgraph_TextDisplay.Name = "Text Display";
    Subgraph_TextDisplay.LineWidth = 12;
    Subgraph_TextDisplay.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
    Subgraph_TextDisplay.PrimaryColor = RGB(0, 0, 0); //black
    Subgraph_TextDisplay.SecondaryColor = RGB(128, 255, 255);
    Subgraph_TextDisplay.SecondaryColorUsed = true;
    Subgraph_TextDisplay.DisplayNameValueInWindowsFlags = 0;

    Input_TextHorizontalPosition.Name.Format("Text - Horizontal Position From Left (1-%d)", static_cast<int>
(CHART_DRAWING_MAX_HORIZONTAL_AXIS_RELATIVE_POSITION));
    Input_TextHorizontalPosition.SetInt(20);
    Input_TextHorizontalPosition.SetIntLimits(1, static_cast<int>
(CHART_DRAWING_MAX_HORIZONTAL_AXIS_RELATIVE_POSITION));

    Input_TextVerticalPosition.Name.Format("Text - Vertical Position From Bottom (1-%d)", static_cast<int>
(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));
    Input_TextVerticalPosition.SetInt(90);
    Input_TextVerticalPosition.SetIntLimits(1, static_cast<int>
(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));

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

    Input_AutoSetLimitPrice.Name = "Auto Set Limit Price";
    Input_AutoSetLimitPrice.SetYesNo(true);

    Input_LimitPriceOffsetInTicks.Name = "Limit Price Offset In Ticks";
    Input_LimitPriceOffsetInTicks.SetInt(4);

    Subgraph_OrderLine.Name = "Order Line";
    Subgraph_OrderLine.PrimaryColor = COLOR_GREEN;
    Subgraph_OrderLine.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_OrderLine.DrawZeros = false;

    Subgraph_TriggerLine.Name = "Trigger Line";
    Subgraph_TriggerLine.PrimaryColor = COLOR_CYAN;
    Subgraph_TriggerLine.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_TriggerLine.DrawZeros = 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;
    sc.MaximumPositionAllowed = 100000000; //use a high-value to effectively not put any Position Quantity restrictions

    return;
}

```

```

int& r_SetTriggerPriceMenuID = sc.GetPersistentInt(1);
int& r_SetSellOrderPriceMenuID = sc.GetPersistentInt(2);
int& r_SetBuyOrderPriceMenuID = sc.GetPersistentInt(3);
int& r_CancelOrderMenuID = sc.GetPersistentInt(4);
int& r_OrderType = sc.GetPersistentInt(5);
int& r_LastSubgraphUpdateIndex = sc.GetPersistentInt(6);
int64_t& r_LastTimeAndSalesRecordSequence = sc.GetPersistentInt64(7);
int& r_OrderModifyMode = sc.GetPersistentInt(8);
int& r_LineNumber = sc.GetPersistentInt(9);

double& r_TriggerPrice = sc.GetPersistentDouble(9);
double& r_OrderPrice = sc.GetPersistentDouble(10);

// Cleanup when study instance is removed
if (sc.LastCallToFunction)
{
    // Be sure to remove the menu commands when study is removed

    if (r_SetTriggerPriceMenuID != 0)
        sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_SetTriggerPriceMenuID);

    if (r_SetBuyOrderPriceMenuID != 0)
        sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_SetBuyOrderPriceMenuID);

    if (r_SetSellOrderPriceMenuID != 0)
        sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_SetSellOrderPriceMenuID);

    if (r_CancelOrderMenuID != 0)
        sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID);

    return;
}

sc.SendOrdersToTradeService = !sc.GlobalTradeSimulationIsOn;

//Initialization
if (sc.UpdateStartIndex == 0)
{
    if (r_SetTriggerPriceMenuID == 0)
        r_SetTriggerPriceMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Set Trigger Price");

    r_LastTimeAndSalesRecordSequence = 0;
    r_LastSubgraphUpdateIndex = 0;

    TradingTriggeredLimitOrderEntry_UpdateTriggerAndOrderPriceTextDisplay(sc);
}

//Handle the 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 the menu event for setting the trigger price
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 (Input_AutoSetLimitPrice.GetYesNo())
{
    if (sc.ChartTradingOrderPrice >= CurrentPrice) // Buy
    {
        r_OrderPrice = sc.ChartTradingOrderPrice - Input_LimitPriceOffsetInTicks.GetInt() * sc.TickSize;
        r_OrderType = 1;
        Subgraph_OrderLine.Name = "Buy Limit";
    }
    else // Sell
    {
        r_OrderPrice = sc.ChartTradingOrderPrice + Input_LimitPriceOffsetInTicks.GetInt() * sc.TickSize;
        r_OrderType = -1;
        Subgraph_OrderLine.Name = "Sell Limit";
    }
}
else
{
    // Add Sell/Buy Order Price menu items
    if (r_SetSellOrderPriceMenuID == 0)
        r_SetSellOrderPriceMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Set Sell Order Price");

    if (r_SetBuyOrderPriceMenuID == 0)
        r_SetBuyOrderPriceMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Set Buy Order Price");
}

if (r_CancelOrderMenuID == 0)
    r_CancelOrderMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Cancel Triggered Limit Order");

r_LastTimeAndSalesRecordSequence = 0;
}

// handle set Sell order menu event
if (sc.MenuEventID != 0 && sc.MenuEventID == r_SetSellOrderPriceMenuID)
{
    //Get the order price
    double NewOrderPrice = sc.ChartTradingOrderPrice;

    // order price must be greater than trigger price for sell orders
    if (NewOrderPrice < r_TriggerPrice)
        sc.AddMessageToLog("Sell Order Price must be greater than the Trigger Price", 1);
    else
    {
        r_LastTimeAndSalesRecordSequence = 0;
        r_OrderPrice = NewOrderPrice;
        r_OrderType = -1;
    }
}

// handle set Buy order menu event
if (sc.MenuEventID != 0 && sc.MenuEventID == r_SetBuyOrderPriceMenuID)
{
    //Get the order price
    double NewOrderPrice = sc.ChartTradingOrderPrice;

    // order price must be less than trigger price for buy orders
    if (NewOrderPrice > r_TriggerPrice)
        sc.AddMessageToLog("Buy Order price must be less than the Trigger Price", 1);
    else
    {
        r_LastTimeAndSalesRecordSequence = 0;
    }
}

```

```

        r_OrderPrice = NewOrderPrice;
        r_OrderType = 1;
    }
}

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

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

    if (r_SetBuyOrderPriceMenuID)
    {
        sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_SetBuyOrderPriceMenuID);
        r_SetBuyOrderPriceMenuID = 0;
    }

    if (r_SetSellOrderPriceMenuID)
    {
        sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_SetSellOrderPriceMenuID);
        r_SetSellOrderPriceMenuID = 0;
    }
}

//Handle order modifications
if (r_OrderType != 0)
{
    if (sc.PointerEventType == SC_POINTER_BUTTON_DOWN
        && r_TriggerPrice >= sc.ChartTradingOrderPrice - sc.TickSize
        && r_TriggerPrice <= 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)
    {
        double TriggerPriceOrderPriceDifference = r_TriggerPrice - r_OrderPrice;
        r_TriggerPrice = sc.ChartTradingOrderPrice;
        r_OrderPrice = r_TriggerPrice - TriggerPriceOrderPriceDifference;
        TradingTriggeredLimitOrderEntry_UpdateTriggerAndOrderPriceTextDisplay(sc);
    }
}

if (sc.MenuEventID != 0)
    TradingTriggeredLimitOrderEntry_UpdateTriggerAndOrderPriceTextDisplay(sc);

//Update the Subgraph lines
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);
}

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

// If the order type is zero, there is nothing to do. Also do not do order evaluation when modifying the order.
if (r_OrderType == 0 || r_OrderModifyMode != 0)
    return;

//Iterate through new time and sales data
c_SCTimeAndSalesArray TimeSalesArray;
sc.GetTimeAndSales(TimeSalesArray);
int StartingIndex = 0;

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

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

bool PerformReset = false;

for (int Index = StartingIndex; Index < TimeSalesArray.Size(); ++Index)
{
    s_TimeAndSales Record = TimeSalesArray[Index];
    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 sell order trigger conditions
    if (r_OrderType == -1
        && sc.FormattedEvaluate(Record.Price, sc.BaseGraphValueFormat, LESS_EQUAL_OPERATOR,
static_cast<float>(r_TriggerPrice), sc.BaseGraphValueFormat))
    {
        s_SCNewOrder Order;
        Order.Price1 = r_OrderPrice;
        Order.OrderType = SCT_ORDERTYPE_LIMIT;
        Order.OrderQuantity = sc.TradeWindowOrderQuantity;

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

```

```

// check for buy order trigger conditions
if (r_OrderType == 1
    && sc.FormattedEvaluate(Record.Price, sc.BaseGraphValueFormat, GREATER_EQUAL_OPERATOR,
static_cast<float>(r_TriggerPrice), sc.BaseGraphValueFormat))
{

    s_SCNewOrder Order;
    Order.Price1 = r_OrderPrice;
    Order.OrderType = SCT_ORDERTYPE_LIMIT;
    Order.OrderQuantity = sc.TradeWindowOrderQuantity;

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

    }

    PerformReset = true;
    break;
}
}

if (PerformReset)
{
    if(r_SetBuyOrderPriceMenuID != 0)
        sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_SetBuyOrderPriceMenuID);

    if(r_SetSellOrderPriceMenuID != 0)
        sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_SetSellOrderPriceMenuID);

    if(r_CancelOrderMenuID != 0)
        sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID);

    r_SetBuyOrderPriceMenuID = 0;
    r_SetSellOrderPriceMenuID = 0;
    r_CancelOrderMenuID = 0;

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

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

    TradingTriggeredLimitOrderEntry_UpdateTriggerAndOrderPriceTextDisplay(sc);
}
}

/*=====*/
SCSFExport scsf_TradingTriggeredMarketOrderForTradeSymbol(SCStudyInterfaceRef sc)
{

    SCSubgraphRef Subgraph_TriggerLine = sc.Subgraph[0];
    SCSubgraphRef Subgraph_TextDisplay = sc.Subgraph[1];

    SCInputRef Input_TextHorizontalPosition = sc.Input[0];
    SCInputRef Input_TextVerticalPosition = sc.Input[1];
    SCInputRef Input_NumberOfBarsBack = sc.Input[2];

    if (sc.SetDefaults)

```

```

{
    sc.GraphName = "Trading: Triggered Market Order for Trade Symbol";
    sc.AutoLoop = 0; //Manual looping
    sc.GraphRegion = 0;
    sc.ScaleRangeType = SCALE_SAMEASREGION;

    Subgraph_TextDisplay.Name = "Text Display";
    Subgraph_TextDisplay.LineWidth = 12;
    Subgraph_TextDisplay.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
    Subgraph_TextDisplay.PrimaryColor = RGB(0, 0, 0); //black
    Subgraph_TextDisplay.SecondaryColor = RGB(128, 255, 255);
    Subgraph_TextDisplay.SecondaryColorUsed = true;
    Subgraph_TextDisplay.DisplayNameValueInWindowsFlags = 0;

    Input_TextHorizontalPosition.Name.Format("Text - Horizontal Position From Left (1-%d)", static_cast<int>
(CHART_DRAWING_MAX_HORIZONTAL_AXIS_RELATIVE_POSITION));
    Input_TextHorizontalPosition.SetInt(20);
    Input_TextHorizontalPosition.SetIntLimits(1, static_cast<int>
(CHART_DRAWING_MAX_HORIZONTAL_AXIS_RELATIVE_POSITION));

    Input_TextVerticalPosition.Name.Format("Text - Vertical Position From Bottom (1-%d)", static_cast<int>
(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));
    Input_TextVerticalPosition.SetInt(90);
    Input_TextVerticalPosition.SetIntLimits(1, static_cast<int>
(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));

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

    Subgraph_TriggerLine.Name = "Trigger Line";
    Subgraph_TriggerLine.PrimaryColor = COLOR_CYAN;
    Subgraph_TriggerLine.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_TriggerLine.DrawZeros = 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;
    sc.MaximumPositionAllowed = 100000000; //use a high-value to effectively not put any Position Quantity restrictions

    return;
}

int& r_SetBuyOrderTriggerMenuID = sc.GetPersistentInt(1);
int& r_SetSellOrderTriggerMenuID = sc.GetPersistentInt(2);

int& r_CancelOrderMenuID = sc.GetPersistentInt(3);
int& r_OrderType = sc.GetPersistentInt(4);
int& r_LastSubgraphUpdateIndex = sc.GetPersistentInt(5);
int64_t& r_LastTimeAndSalesRecordSequence = sc.GetPersistentInt64(6);
int& r_OrderModifyMode = sc.GetPersistentInt(7);
int& r_LineNumber = sc.GetPersistentInt(8);

double& r_TriggerPrice = sc.GetPersistentDouble(9);

// Cleanup when study instance is removed
if (sc.LastCallToFunction)

```

```

{
    // Be sure to remove the menu commands when study is removed

    if (r_SetBuyOrderTriggerMenuID != 0)
        sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_SetBuyOrderTriggerMenuID);

    if (r_SetSellOrderTriggerMenuID != 0)
        sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_SetSellOrderTriggerMenuID);

    if (r_CancelOrderMenuID != 0)
        sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID);

    return;
}

sc.SendOrdersToTradeService = !sc.GlobalTradeSimulationIsOn;

//Initialization
if (sc.UpdateStartIndex == 0)
{
    // Add Sell/Buy Order Price menu items
    if (r_SetSellOrderTriggerMenuID == 0)
        r_SetSellOrderTriggerMenuID = sc.AddACSChartShortcutMenuItem(sc.ChartNumber, "Set Sell Order Trigger");

    if (r_SetBuyOrderTriggerMenuID == 0)
        r_SetBuyOrderTriggerMenuID = sc.AddACSChartShortcutMenuItem(sc.ChartNumber, "Set Buy Order Trigger");

    r_LastTimeAndSalesRecordSequence = 0;
    r_LastSubgraphUpdateIndex = 0;
}

//Handle the 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 Sell order menu event
if (sc.MenuEventID != 0 && sc.MenuEventID == r_SetSellOrderTriggerMenuID)
{
    //Get the order price
    r_TriggerPrice = sc.ChartTradingOrderPrice;

    r_LastTimeAndSalesRecordSequence = 0;
    r_OrderType = -1;

    if (r_CancelOrderMenuID == 0)
        r_CancelOrderMenuID = sc.AddACSChartShortcutMenuItem(sc.ChartNumber, "Cancel Trigger");
}

// handle set Buy order menu event
if (sc.MenuEventID != 0 && sc.MenuEventID == r_SetBuyOrderTriggerMenuID)
{
    //Get the order price
    r_TriggerPrice = sc.ChartTradingOrderPrice;

```

```

r_LastTimeAndSalesRecordSequence = 0;
r_OrderType = 1;

if (r_CancelOrderMenuID == 0)
    r_CancelOrderMenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Cancel Trigger");

}

// handle cancel order menu event
if (sc.MenuEventID != 0 && sc.MenuEventID == r_CancelOrderMenuID)
{
    r_OrderType = 0;
    r_TriggerPrice = 0.0;
    r_LastTimeAndSalesRecordSequence = 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_TriggerPrice >= sc.ChartTradingOrderPrice - sc.TickSize
        && r_TriggerPrice <= 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_TriggerPrice = sc.ChartTradingOrderPrice;
    }

}

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

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

r_LastSubgraphUpdateIndex = StartingOutputIndex;

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

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

// If the order type is zero, there is nothing to do. Also do not to order evaluation when modifying the order.

```

```

if (r_OrderType == 0 || r_OrderModifyMode != 0)
    return;

//Iterate through new time and sales data
c_SCTimeAndSalesArray TimeSalesArray;
sc.GetTimeAndSales(TimeSalesArray);
int StartingIndex = 0;

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

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

bool PerformReset = false;

for (int Index = StartingIndex; Index < TimeSalesArray.Size(); ++Index)
{
    s_TimeAndSales Record = TimeSalesArray[Index];
    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 sell order trigger conditions
    if (r_OrderType == -1
        && sc.FormattedEvaluate(Record.Price, sc.BaseGraphValueFormat, GREATER_EQUAL_OPERATOR,
static_cast<float>(r_TriggerPrice), 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;
            break;
        }

        // check for buy order trigger conditions
        if (r_OrderType == 1
            && sc.FormattedEvaluate(Record.Price, sc.BaseGraphValueFormat, LESS_EQUAL_OPERATOR,
static_cast<float>(r_TriggerPrice), 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;
    break;
}
}

if (PerformReset)
{
    if (r_CancelOrderMenuID != 0)
        sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_CancelOrderMenuID);

    r_CancelOrderMenuID = 0;

    r_TriggerPrice = 0.0;
    r_OrderType = 0;
    r_LastTimeAndSalesRecordSequence = 0;

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

/*=====*/

```