

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

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

```
SCSFExport scsf_TradingExample(SCStudyInterfaceRef sc)
```

```
{  
    //Define references to the Subgraphs and Inputs for easy reference  
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];  
    SCSubgraphRef Subgraph_BuyExit = sc.Subgraph[1];  
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];  
    SCSubgraphRef Subgraph_SellExit = sc.Subgraph[3];  
  
    SCSubgraphRef Subgraph_SimpMovAvg = sc.Subgraph[4];  
  
    SCInputRef Input_Enabled = sc.Input[0];  
    SCInputRef Input_TargetValue = sc.Input[1];  
    SCInputRef Input_StopValue = sc.Input[2];  
  
    if (sc.SetDefaults)  
    {  
        // Set the study configuration and defaults.  
  
        sc.GraphName = "Trading Example: Using Moving Average and Target and Stop";  
  
        Subgraph_BuyEntry.Name = "Buy Entry";  
        Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;  
        Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);  
        Subgraph_BuyEntry.LineWidth = 2;  
        Subgraph_BuyEntry.DrawZeros = false;  
  
        Subgraph_BuyExit.Name = "Buy Exit";  
        Subgraph_BuyExit.DrawStyle = DRAWSTYLE_ARROW_DOWN;  
        Subgraph_BuyExit.PrimaryColor = RGB(255, 128, 128);  
        Subgraph_BuyExit.LineWidth = 2;  
        Subgraph_BuyExit.DrawZeros = false;  
  
        Subgraph_SellEntry.Name = "Sell Entry";  
        Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;  
        Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);  
        Subgraph_SellEntry.LineWidth = 2;  
        Subgraph_SellEntry.DrawZeros = false;  
  
        Subgraph_SellExit.Name = "Sell Exit";  
        Subgraph_SellExit.DrawStyle = DRAWSTYLE_ARROW_UP;  
        Subgraph_SellExit.PrimaryColor = RGB(128, 255, 128);  
        Subgraph_SellExit.LineWidth = 2;  
        Subgraph_SellExit.DrawZeros = false;  
  
        Subgraph_SimpMovAvg.Name = "Simple Moving Average";  
        Subgraph_SimpMovAvg.DrawStyle = DRAWSTYLE_LINE;  
        Subgraph_SimpMovAvg.DrawZeros = false;  
  
        Input_Enabled.Name = "Enabled";  
        Input_Enabled.SetYesNo(0);  
  
        Input_TargetValue.Name = "Target Value";  
        Input_TargetValue.SetFloat(2.0f);  
  
        Input_StopValue.Name = "Stop Value";  
        Input_StopValue.SetFloat(1.0f);  
    }  
}
```

sc.StudyDescription = "This study function is an example of how to use the ACSIL Trading Functions. This function will display a simple moving average and perform a Buy Entry when the Last price crosses the moving average from below and a Sell Entry when the Last price crosses the moving average from above. A new entry cannot occur until the Target or Stop has been hit. When an order is sent, a corresponding arrow will appear on the chart to show that an order

was sent. This study will do nothing until the Enabled Input is set to Yes.";

```
sc.AutoLoop = 1;
sc.GraphRegion = 0;

//Any of the following variables can also be set outside and below the sc.SetDefaults code block

sc.AllowMultipleEntriesInSameDirection = false;
sc.MaximumPositionAllowed = 10;
sc.SupportReversals = false;

// This is false by default. Orders will go to the simulation system always.
sc.SendOrdersToTradeService = false;

sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;
sc.SupportAttachedOrdersForTrading = false;
sc.CancelAllOrdersOnEntriesAndReversals= true;
sc.AllowEntryWithWorkingOrders = false;
sc.CancelAllWorkingOrdersOnExit = false;

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

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

return;
}

if (!Input_Enabled.GetYesNo())
return;

SCFloatArrayRef Last = sc.Close;

// Calculate the moving average
sc.SimpleMovAvg(Last, Subgraph_SimpMovAvg, sc.Index, 10);

if (sc.IsFullRecalculation)
return;

// Get the Trade Position data to be used for position exit processing.
s_SCPositionData PositionData;
sc.GetTradePosition(PositionData) ;

float LastTradePrice = sc.Close[sc.Index];

int64_t& r_BuyEntryInternalOrderID = sc.GetPersistentInt64(1);

// Create an s_SCNewOrder object.
s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 1;
NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
NewOrder.TextTag = "Trading example tag";
//NewOrder.Symbol = "Test";
NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;

int Result = 0;

//Optional: Check if within allowed time range. In this example we will use 10:00 through 14:00. This is according to the
time zone of the chart.
//int BarTime = sc.BaseDateTimeIn[sc.Index].GetTime();
//bool TradingAllowed = BarTime >= HMS_TIME(10, 0, 0) && BarTime < HMS_TIME(14, 0, 0);
```

```

bool TradingAllowed = true;

// Buy when the last price crosses the moving average from below.
if (TradingAllowed && sc.CrossOver>Last, Subgraph_SimpMovAvg) == CROSS_FROM_BOTTOM &&
sc.GetBarHasClosedStatus() == BHCS_BAR_HAS_CLOSED)
{
    Result = static_cast<int>(sc.BuyEntry(NewOrder));
    //Result = sc.BuyOrder(NewOrder);
    //If there has been a successful order entry, then draw an arrow at the low of the bar.
    if (Result > 0)
    {
        r_BuyEntryInternalOrderID = NewOrder.InternalOrderID;
        SCString InternalOrderIDNumberString;
        InternalOrderIDNumberString.Format("BuyEntry Internal Order ID: %d", r_BuyEntryInternalOrderID);
        sc.AddMessageToLog(InternalOrderIDNumberString, 0);

        Subgraph_BuyEntry[sc.Index] = sc.Low[sc.Index];
    }
}

// When there is a long position AND the Last price is less than the price the Buy Entry was filled at minus Stop Value,
OR there is a long position AND the Last price is greater than the price the Buy Entry was filled at plus the Target Value.
else if (PositionData.PositionQuantity > 0
    && (LastTradePrice <= PositionData.AveragePrice - Input_StopValue.GetFloat() ||
    LastTradePrice >= PositionData.AveragePrice + Input_TargetValue.GetFloat()))
{
    Result = static_cast<int>(sc.BuyExit(NewOrder));

    //If there has been a successful order entry, then draw an arrow at the high of the bar.
    if (Result > 0)
    {
        Subgraph_BuyExit[sc.Index] = sc.High[sc.Index];
    }
}

// Sell when the last price crosses the moving average from above.
else if (TradingAllowed && sc.CrossOver>Last, Subgraph_SimpMovAvg) == CROSS_FROM_TOP &&
sc.GetBarHasClosedStatus() == BHCS_BAR_HAS_CLOSED)
{
    Result = static_cast<int>(sc.SellEntry(NewOrder));

    // If there has been a successful order entry, then draw an arrow at the high of the bar.
    if (Result > 0)
    {
        Subgraph_SellEntry[sc.Index] = sc.High[sc.Index];
    }
}

// When there is a short position AND the Last price is greater than the price the Sell Entry was filled at plus the Stop
Value, OR there is a short position AND the Last price is less than the price the Sell Entry was filled at minus the Target
Value.
else if (PositionData.PositionQuantity < 0
    && (LastTradePrice >= PositionData.AveragePrice + Input_StopValue.GetFloat() ||
    LastTradePrice <= PositionData.AveragePrice - Input_TargetValue.GetFloat()))
{
    Result = static_cast<int>(sc.SellExit(NewOrder));

    // If there has been a successful order entry, then draw an arrow at the low of the bar.
    if (Result > 0)
    {
        Subgraph_SellExit[sc.Index] = sc.Low[sc.Index];
    }
}

```

```

}

//Example to check the status of the order using the Internal Order ID remembered in a reference to a persistent
variable.
if (r_BuyEntryInternalOrderID != 0)
{
    s_SCTradeOrder TradeOrder;
    sc.GetOrderByOrderID(r_BuyEntryInternalOrderID, TradeOrder);

    // Order has been found.
    if (TradeOrder.InternalOrderID == r_BuyEntryInternalOrderID)
    {
        bool IsOrderFilled = TradeOrder.OrderStatusCode == SCT_OSC_FILLED;
        double FillPrice = TradeOrder.LastFillPrice;
    }
}
}

/*=====*/
SCSFExport scsf_TradingExampleWithAttachedOrders(SCStudyInterfaceRef sc)
{
    //Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SimpMovAvg = sc.Subgraph[4];

    SCInputRef Input_Enabled = sc.Input[0];

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

        sc.GraphName = "Trading Example: With Trade Window Attached Orders";

        Subgraph_BuyEntry.Name = "Buy Entry";
        Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
        Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
        Subgraph_BuyEntry.LineWidth = 2;
        Subgraph_BuyEntry.DrawZeros = false;

        Subgraph_SellEntry.Name = "Sell Entry";
        Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
        Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
        Subgraph_SellEntry.LineWidth = 2;
        Subgraph_SellEntry.DrawZeros = false;

        Subgraph_SimpMovAvg.Name = "Simple Moving Average";
        Subgraph_SimpMovAvg.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SimpMovAvg.DrawZeros = false;

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

        sc.StudyDescription = "This study function is an example of how to use the ACSIL Trading Functions. This example
uses the Attached Orders defined on the chart Trade Window. This function will display a simple moving average and
perform a Buy Entry when the Last price crosses the moving average from below and a Sell Entry when the Last price
crosses the moving average from above. A new entry cannot occur until the Target or Stop has been hit. When an order is
sent, a corresponding arrow will appear on the chart to show that an order was sent. This study will do nothing until the
Enabled Input is set to Yes.";

        sc.AutoLoop = 1;
        sc.GraphRegion = 0;

        //Any of the following variables can also be set outside and below the sc.SetDefaults code block

```

```

sc.AllowMultipleEntriesInSameDirection = false;
sc.MaximumPositionAllowed = 10;
sc.SupportReversals = false;

// This is false by default. Orders will go to the simulation system always.
sc.SendOrdersToTradeService = false;

sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;
sc.SupportAttachedOrdersForTrading = true;

// This line can be within sc.SetDefaults or outside of it.
//sc.TradeWindowConfigFileName = "Test_2.twconfig";

sc.CancelAllOrdersOnEntriesAndReversals= true;
sc.AllowEntryWithWorkingOrders = false;
sc.CancelAllWorkingOrdersOnExit = true;

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

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

return;
}

if (!Input_Enabled.GetYesNo())
return;

// Calculate the moving average
sc.SimpleMovAvg(sc.Close, Subgraph_SimpMovAvg, sc.Index, 10);

if (sc.IsFullRecalculation)
return;

// Create an s_SCNewOrder object.
s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 1;
NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;

// Buy when the last price crosses the moving average from below.
if (sc.CrossOver(sc.Close, Subgraph_SimpMovAvg) == CROSS_FROM_BOTTOM && sc.GetBarHasClosedStatus() ==
BHCS_BAR_HAS_CLOSED)
{
//Buy Entry. Attached orders defined on Trade Window will be used.
int Result = static_cast<int>(sc.BuyEntry(NewOrder));
if (Result > 0) //If there has been a successful order entry, then draw an arrow at the low of the bar.
Subgraph_BuyEntry[sc.Index] = sc.Low[sc.Index];
}

// Sell when the last price crosses the moving average from above.
else if (sc.CrossOver(sc.Close, Subgraph_SimpMovAvg) == CROSS_FROM_TOP && sc.GetBarHasClosedStatus() ==
BHCS_BAR_HAS_CLOSED)
{
//Sell Entry. Attached orders defined on Trade Window will be used.
int Result = static_cast<int>(sc.SellEntry(NewOrder));
if (Result > 0) //If there has been a successful order entry, then draw an arrow at the high of the bar.
Subgraph_SellEntry[sc.Index] = sc.High[sc.Index];
}
}

```

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

```
SCSFExport scsf_TradingExampleWithAttachedOrdersDirectlyDefined(SCStudyInterfaceRef sc)
```

```
{  
    // Define references to the Subgraphs and Inputs for easy reference  
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];  
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];  
    SCSubgraphRef Subgraph_SimpMovAvg = sc.Subgraph[4];  
  
    SCInputRef Enabled = sc.Input[0];  
  
    if (sc.SetDefaults)  
    {  
        // Set the study configuration and defaults.  
  
        sc.GraphName = "Trading Example: With Hardcoded Attached Orders";  
  
        Subgraph_BuyEntry.Name = "Buy Entry";  
        Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;  
        Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);  
        Subgraph_BuyEntry.LineWidth = 2;  
        Subgraph_BuyEntry.DrawZeros = false;  
  
        Subgraph_SellEntry.Name = "Sell Entry";  
        Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;  
        Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);  
        Subgraph_SellEntry.LineWidth = 2;  
        Subgraph_SellEntry.DrawZeros = false;  
  
        Subgraph_SimpMovAvg.Name = "Simple Moving Average";  
        Subgraph_SimpMovAvg.DrawStyle = DRAWSTYLE_LINE;  
        Subgraph_SimpMovAvg.DrawZeros = false;  
  
        Enabled.Name = "Enabled";  
        Enabled.SetYesNo(0);  
  
        sc.StudyDescription = "This study function is an example of how to use the ACSIL Trading Functions. This example  
uses Attached Orders defined directly within this function. This function will display a simple moving average and perform  
a Buy Entry when the Last price crosses the moving average from below and a Sell Entry when the Last price crosses the  
moving average from above. A new entry cannot occur until the Target or Stop has been hit. When an order is sent, a  
corresponding arrow will appear on the chart to show that an order was sent. This study will do nothing until the Enabled  
Input is set to Yes.";  
  
        sc.AllowMultipleEntriesInSameDirection = false;  
        sc.MaximumPositionAllowed = 5;  
        sc.SupportReversals = false;  
  
        // This is false by default. Orders will go to the simulation system always.  
        sc.SendOrdersToTradeService = false;  
  
        sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;  
  
        // This can be false in this function because we specify Attached Orders directly with the order which causes this to  
be considered true when submitting an order.  
        sc.SupportAttachedOrdersForTrading = false;  
  
        sc.CancelAllOrdersOnEntriesAndReversals= true;  
        sc.AllowEntryWithWorkingOrders = false;  
        sc.CancelAllWorkingOrdersOnExit = true;  
  
        // Only 1 trade for each Order Action type is allowed per bar.  
        sc.AllowOnlyOneTradePerBar = true;  
  
        //This needs to be set to true when a trading study uses trading functions.  
        sc.MaintainTradeStatisticsAndTradesData = true;  
    }  
}
```

```

sc.AutoLoop = 1;
sc.GraphRegion = 0;

return;
}

if (!Enabled.GetYesNo())
return;

// Use persistent variables to remember attached order IDs so they can be modified or canceled.
int32_t& Target1OrderID = sc.GetPersistentInt(1);
int32_t& Stop1OrderID = sc.GetPersistentInt(2);

// Calculate the moving average
sc.SimpleMovAvg(sc.Close, Subgraph_SimpMovAvg, sc.Index, 10);

if (sc.IsFullRecalculation)
return;

// Create an s_SCNewOrder object.
s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 1;
NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;

//Specify a Target and a Stop with 8 tick offsets. We are specifying one Target and one Stop, additional targets and
stops can be specified as well.
NewOrder.Target1Offset = 8 * sc.TickSize;
NewOrder.Stop1Offset = 8 * sc.TickSize;
NewOrder.OCOGroup1Quantity = 1; // If this is left at the default of 0, then it will be automatically set.

// Buy when the last price crosses the moving average from below.
if (sc.CrossOver(sc.Close, Subgraph_SimpMovAvg) == CROSS_FROM_BOTTOM && sc.GetBarHasClosedStatus() ==
BHCS_BAR_HAS_CLOSED)
{
int Result = static_cast<int>(sc.BuyEntry(NewOrder));
if (Result > 0) //If there has been a successful order entry, then draw an arrow at the low of the bar.
{
Subgraph_BuyEntry[sc.Index] = sc.Low[sc.Index];

// Remember the order IDs for subsequent modification and cancellation
Target1OrderID = NewOrder.Target1InternalOrderID;
Stop1OrderID = NewOrder.Stop1InternalOrderID;
}
}

// Sell when the last price crosses the moving average from above.
else if (sc.CrossOver(sc.Close, Subgraph_SimpMovAvg) == CROSS_FROM_TOP && sc.GetBarHasClosedStatus() ==
BHCS_BAR_HAS_CLOSED)
{
int Result = static_cast<int>(sc.SellEntry(NewOrder));
if (Result > 0) //If there has been a successful order entry, then draw an arrow at the high of the bar.
{
Subgraph_SellEntry[sc.Index] = sc.High[sc.Index];

// Remember the order IDs for subsequent modification and cancellation
Target1OrderID = NewOrder.Target1InternalOrderID;
Stop1OrderID = NewOrder.Stop1InternalOrderID;
}
}

// This code block serves as an example of how to modify an attached order. In this example it is set to false and never

```

runs.

```
bool ExecuteModifyOrder = false;
if (ExecuteModifyOrder && (sc.Index == sc.ArraySize - 1))//Only do a modify on the most recent bar
{
    double NewLimit = 0.0;

    // Get the existing target order
    s_SCTradeOrder ExistingOrder;
    if (sc.GetOrderByOrderID(Target1OrderID, ExistingOrder) != SCTRADING_ORDER_ERROR)
    {
        if (ExistingOrder.BuySell == BSE_BUY)
            NewLimit = sc.Close[sc.Index] - 5*sc.TickSize;
        else if (ExistingOrder.BuySell == BSE_SELL)
            NewLimit = sc.Close[sc.Index] + 5*sc.TickSize;

        // We can only modify price and/or quantity

        s_SCNewOrder ModifyOrder;
        ModifyOrder.InternalOrderID = Target1OrderID;
        ModifyOrder.Price1 = NewLimit;

        sc.ModifyOrder(ModifyOrder);
    }
}

////Target 1
//NewOrder.Target1Offset = 10*sc.TickSize;
//NewOrder.AttachedOrderTarget1Type = SCT_ORDERTYPE_LIMIT;

////Target 2
//NewOrder.Target2Offset = 10*sc.TickSize;
//NewOrder.AttachedOrderTarget2Type = SCT_ORDERTYPE_LIMIT;

////Common Stop
//NewOrder.StopAllOffset = 10* sc.TickSize;
//NewOrder.AttachedOrderStopAllType = SCT_ORDERTYPE_STOP;
}

/*=====*/
//This function is used for ACSIL trading testing. It does *not* serve as an example.

SCSFExport scsf_ACSILTradingTest(SCStudyInterfaceRef sc)
{
    //Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BuyExit = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SellExit = sc.Subgraph[3];

    SCInputRef Input_Enabled = sc.Input[0];

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

        sc.GraphName = "Trading Test";

        Subgraph_BuyEntry.Name = "Buy Entry";
        Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
        Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
        Subgraph_BuyEntry.LineWidth = 2;
        Subgraph_BuyEntry.DrawZeros = false;
```



```

Subgraph_BuyExit.Name = "Buy Exit";
Subgraph_BuyExit.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_BuyExit.PrimaryColor = RGB(255, 128, 128);
Subgraph_BuyExit.LineWidth = 2;
Subgraph_BuyExit.DrawZeros = false;

Subgraph_SellEntry.Name = "Sell Entry";
Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
Subgraph_SellEntry.LineWidth = 2;
Subgraph_SellEntry.DrawZeros = false;

Subgraph_SellExit.Name = "Sell Exit";
Subgraph_SellExit.DrawStyle = DRAWSTYLE_ARROW_UP;
Subgraph_SellExit.PrimaryColor = RGB(128, 255, 128);
Subgraph_SellExit.LineWidth = 2;
Subgraph_SellExit.DrawZeros = false;

Input_Enabled.Name = "Enabled";
Input_Enabled.SetYesNo(0);
Input_Enabled. SetDescription("This input enables the study and allows it to function. Otherwise, it does nothing.");

// This is false by default. Orders will be simulated.
sc.SendOrdersToTradeService = false;

sc.AllowMultipleEntriesInSameDirection = false;

//This must be equal to or greater than the order quantities you will be submitting orders for.
sc.MaximumPositionAllowed = 10000;

sc.SupportReversals = true;
sc.AllowOppositeEntryWithOpposingPositionOrOrders = true;

// This variable controls whether to use or not use attached orders that are configured on the Trade Window for the
chart.
sc.SupportAttachedOrdersForTrading = false;

//This variable controls whether to use the "Use Attached Orders" setting on the Trade Window for the chart
sc.UseGUIAttachedOrderSetting = false;

sc.CancelAllOrdersOnEntriesAndReversals = false;
sc.AllowEntryWithWorkingOrders = true;
sc.CancelAllWorkingOrdersOnExit = true;
sc.AllowOnlyOneTradePerBar = false;

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

sc.AutoLoop = true;
sc.GraphRegion = 0;

return;
}

//These must be outside of the sc.SetDefaults code block since they have a dependency upon an actual chart object
existing
sc.SupportTradingScaleIn = 0;
sc.SupportTradingScaleOut = 0;

if (!Input_Enabled.GetYesNo())
return;

if (sc.LastCallToFunction)

```

```

    return;//nothing to do

    if (sc.IsFullRecalculation)
        return;

    // Run the below code only on the last bar
    if (sc.Index < sc.ArraySize-1)
        return;

    s_SCPositionData PositionDataForSymbolAndAccount;
    sc.GetTradePositionForSymbolAndAccount(PositionDataForSymbolAndAccount, sc.Symbol,
    sc.SelectedTradeAccount);

    s_SCPositionData PositionData;
    sc.GetTradePosition(PositionData);

    if (PositionData.PositionQuantity == 0 && !PositionData.WorkingOrdersExist)
    {
        //// Create an s_SCNewOrder object.
        //s_SCNewOrder NewOrder;
        //NewOrder.OrderQuantity = 1;
        //NewOrder.OrderType = SCT_ORDERTYPE_STOP_LIMIT;
        //NewOrder.TimeInForce = SCT_TIF_DAY;
        //NewOrder.Price1 = sc.Close[sc.Index] - 3 * sc.TickSize;
        //NewOrder.Price2 = NewOrder.Price1 - 4 * sc.TickSize;

        ////Specify a Target and a Stop with 8 tick offsets. We are specifying one Target and one Stop, additional targets and
        stops can be specified as well.
        ////NewOrder.Target1Offset = 8*sc.TickSize;
        ////NewOrder.Stop1Offset = 8*sc.TickSize;
        ////NewOrder.OCOGroup1Quantity = 1; // If this is left at the default of 0, then it will be automatically set.
        //int Result = (int)sc.SellEntry(NewOrder);

        s_SCNewOrder NewOrder;

        NewOrder.OrderQuantity = 1;
        NewOrder.OrderType = SCT_ORDERTYPE_LIMIT;
        NewOrder.Price1 = sc.Close[sc.Index] - 5 * sc.TickSize;
        NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
        NewOrder.AttachedOrderStopAllType = SCT_ORDERTYPE_TRIGGERED_STEP_TRAILING_STOP;
        // NewOrder.Target1Offset = 15 * sc.TickSize;
        // NewOrder.StopAllOffset = 10 * sc.TickSize;
        // NewOrder.OCOGroup1Quantity = 1;
        //
        // //Trail Offset - Set Correctly in TradeWindow Parameters
        // NewOrder.TriggeredTrailStopTrailPriceOffset = 5 * sc.TickSize;
        // //Trigger Offset - Set Correctly in TradeWindow Parameters
        // NewOrder.TriggeredTrailStopTriggerPriceOffset = 3 * sc.TickSize;
        // //Step Amount - This DOESN't get Set Correctly in TradeWindow parameter
        // NewOrder.TrailStopStepPriceAmount = 1 * sc.TickSize;

        int Result = static_cast<int>(sc.BuyEntry(NewOrder));
    }
    else //if (PositionData.PositionQuantity < 0)
    {
        //// Create an s_SCNewOrder object.
        //s_SCNewOrder NewOrder;
        //NewOrder.OrderQuantity = 1;
        //NewOrder.OrderType = SCT_ORDERTYPE_LIMIT;
        //NewOrder.TimeInForce = SCT_TIF_DAY;

```

```

        //NewOrder.Price1 = sc.Close[sc.Index] - 3 * sc.TickSize;
        ///Specify a Target and a Stop with 8 tick offsets. We are specifying one Target and one Stop, additional targets and
stops can be specified as well.
        //NewOrder.Target1Offset = 8*sc.TickSize;
        //NewOrder.Stop1Offset = 8*sc.TickSize;
        //NewOrder.OCOGroup1Quantity = 1; // If this is left at the default of 0, then it will be automatically set.
        //int Result = (int)sc.BuyEntry(NewOrder);
        double PositionQuantity = PositionData.PositionQuantity;

    }
    //else
    //{
    //    s_SCNewOrder NewOrder;
    //    NewOrder.OrderQuantity = 0; //Flatten
    //    NewOrder.OrderType = SCT_MARKET;
    //    (int)sc.BuyExit(NewOrder);
    //}

    //int &OrderCount = sc.GetPersistentInt(1);

    //if (OrderCount > 0)
    //    return;
    //

    //s_SCNewOrder EntryOrder;
    //EntryOrder.OrderQuantity = 10000; // full position size
    //EntryOrder.OrderType = SCT_MARKET;

    //int Result = (int)sc.BuyEntry(EntryOrder);
    //if(Result > 0)
    //{
    //
    //

    //    s_SCNewOrder ExitOrder;
    //    ExitOrder.OrderQuantity = 10000; // full position size
    //    ExitOrder.OrderType = SCT_MARKET;

    //    int quantity = (int)sc.BuyExit(ExitOrder);
    //    OrderCount++;
    //}

    //
    //    sc.MaximumPositionAllowed += 1;
    //
    //    int &RememberedOrderID= sc.GetPersistentInt(0);
    //    bool CancelOrder = false;
    //
    //    // Create a s_SCNewOrder object.
    //    s_SCNewOrder NewOrder;
    //    NewOrder.OrderQuantity = 1;
    //    NewOrder.OrderType = SCT_MARKET;
    //
    //
    //    int Result =(int)sc.BuyEntry(NewOrder);
    //    if (Result > 0) //If there has been a successful order entry
    //    {
    //        //Remember the order ID for subsequent order modification or cancellation.
    //        RememberedOrderID = NewOrder.InternalOrderID;
    //
    //        //Put an arrow on the chart to indicate the order entry
    //        BuyEntrySubgraph[sc.Index] = sc.Low[sc.Index];
    //    }
    //
    //

```

```

// if(CancelOrder)
// {
//     int Result = sc.CancelOrder(RememberedOrderID);
// }

// Example of submitting an order and handling error condition
// s_SCNewOrder NewOrder;
// NewOrder.OrderQuantity = 1;
// NewOrder.OrderType = SCT_LIMIT;
// NewOrder.TimeInForce = SCT_TIF_DAY;
// NewOrder.Price1 = sc.Close[sc.Index] ;
// int Result = (int)sc.BuyEntry(NewOrder);
// if (Result > 0)//order was accepted
// {
//     //Take appropriate action if order is successful
// }
// else//order error
// {
//     //Only report error if at the last bar
//     if (sc.Index == sc.ArraySize -1)
//     {
//         //Add error message to the Sierra Chart Message Log for interpretation
//         sc.AddMessageToLog(sc.GetTradingErrorMessage(Result), 0);
//     }
// }

}

/*=====*/
SCSFExport scsf_TradingExampleUsingReversals(SCStudyInterfaceRef sc)
{
    // Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];

    SCSubgraphRef Subgraph_SimpMovAvg = sc.Subgraph[4];

    SCInputRef Input_Enabled = sc.Input[0];

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

        sc.GraphName = "Trading Example: Using Reversals";

        Subgraph_BuyEntry.Name = "Buy Entry";
        Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
        Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
        Subgraph_BuyEntry.LineWidth = 2;
        Subgraph_BuyEntry.DrawZeros = false;

        Subgraph_SellEntry.Name = "Sell Entry";
        Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
        Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
        Subgraph_SellEntry.LineWidth = 2;
        Subgraph_SellEntry.DrawZeros = false;

        Subgraph_SimpMovAvg.Name = "Simple Moving Average";
        Subgraph_SimpMovAvg.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SimpMovAvg.DrawZeros = false;

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

```

sc.StudyDescription = "This study function is an example of how to use the ACSIL Trading Functions and demonstrates using reversals. This study will do nothing until the Enabled Input is set to Yes.";

sc.AllowMultipleEntriesInSameDirection = false;

//This must be equal to or greater than the order quantities you will be submitting orders for.

sc.MaximumPositionAllowed = 2;

sc.SupportReversals = true;

// This is false by default. Orders will go to the simulation system always.

sc.SendOrdersToTradeService = false;

sc.AllowOppositeEntryWithOpposingPositionOrOrders = true;

sc.SupportAttachedOrdersForTrading = false;

sc.CancelAllOrdersOnEntriesAndReversals= true;

sc.AllowEntryWithWorkingOrders = false;

sc.CancelAllWorkingOrdersOnExit = true;

sc.AllowOnlyOneTradePerBar = true;

//This needs to be set to true when a trading study uses trading functions.

sc.MaintainTradeStatisticsAndTradesData = true;

sc.AutoLoop = 1;

sc.GraphRegion = 0;

return;

}

if (!Input\_Enabled.GetYesNo())

return;

SCFloatArrayRef BarLastArray = sc.Close;

// Calculate the moving average

sc.SimpleMovAvg(BarLastArray, Subgraph\_SimpMovAvg, sc.Index, 3);

// Create an s\_SCNewOrder object.

s\_SCNewOrder NewOrder;

NewOrder.OrderType = SCT\_ORDERTYPE\_LIMIT;

NewOrder.TimeInForce = SCT\_TIF\_GOOD\_TILL\_CANCELED;

// Buy when the last price of the current bar crosses the moving average from below. This will reverse the position if a short position exists.

if (sc.CrossOver(BarLastArray, Subgraph\_SimpMovAvg) == CROSS\_FROM\_BOTTOM &&  
sc.GetBarHasClosedStatus() == BHCS\_BAR\_HAS\_CLOSED)

{

NewOrder.Price1 = sc.Ask;

NewOrder.OrderQuantity = 1;

int Result = static\_cast<int>(sc.BuyEntry(NewOrder));

if (Result > 0) //If there has been a successful order entry, then draw an arrow at the low of the bar.

{

Subgraph\_BuyEntry[sc.Index] = sc.Low[sc.Index];

}

}

// Sell when the last price of the current bar crosses the moving average from above. This will reverse the position if a long position exists.

else if (sc.CrossOver(BarLastArray, Subgraph\_SimpMovAvg) == CROSS\_FROM\_TOP &&

```

sc.GetBarHasClosedStatus() == BHCS_BAR_HAS_CLOSED)
{
    NewOrder.Price1 = sc.Bid;
    NewOrder.OrderQuantity = 1;
    int Result = static_cast<int>(sc.SellEntry(NewOrder));
    if (Result > 0) //If there has been a successful order entry, then draw an arrow at the high of the bar.
    {
        Subgraph_SellEntry[sc.Index] = sc.High[sc.Index];
    }
}
}
}

```

```

/*=====*/

```

```

SCSFExport scsf_TradingExampleWithAttachedOrdersUsingActualPrices (SCStudyInterfaceRef sc)
{

```

```

    // Define references to the Subgraphs and Inputs for easy reference

```

```

    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SimpMovAvg = sc.Subgraph[4];
    SCInputRef Input_Enabled = sc.Input[0];

```

```

    if (sc.SetDefaults)
    {

```

```

        // Set the study configuration and defaults.

```

```

        sc.GraphName = "Trading Example: With Attached Orders Using Actual Prices";

```

```

        Subgraph_BuyEntry.Name = "Buy Entry";
        Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
        Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
        Subgraph_BuyEntry.LineWidth = 2;
        Subgraph_BuyEntry.DrawZeros = false;

```

```

        Subgraph_SellEntry.Name = "Sell Entry";
        Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
        Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
        Subgraph_SellEntry.LineWidth = 2;
        Subgraph_SellEntry.DrawZeros = false;

```

```

        Subgraph_SimpMovAvg.Name = "Simple Moving Average";
        Subgraph_SimpMovAvg.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SimpMovAvg.DrawZeros = false;

```

```

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

```

sc.StudyDescription = "This study function is an example of how to use the ACSIL Trading Functions. This example uses Attached Orders defined directly within this function. This function will display a simple moving average and perform a Buy Entry when the Last price crosses the moving average from below and a Sell Entry when the Last price crosses the moving average from above. A new entry cannot occur until the Target or Stop has been filled. When an order is sent, a corresponding arrow will appear on the chart to show that an order was sent. This study will do nothing until the Enabled Input is set to Yes.";

```

        sc.AllowMultipleEntriesInSameDirection = false;

```

```

        //This must be equal to or greater than the order quantities you will be submitting orders for.
        sc.MaximumPositionAllowed = 5;

```

```

        sc.SupportReversals = false;

```

```

// This is false by default. Orders will go to the simulation system always.
sc.SendOrdersToTradeService = false;

sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;

// This can be false in this function because we specify Attached Orders directly with the order which causes this to
be considered true when submitting an order.
sc.SupportAttachedOrdersForTrading = false;

sc.CancelAllOrdersOnEntriesAndReversals= true;
sc.AllowEntryWithWorkingOrders = false;
sc.CancelAllWorkingOrdersOnExit = true;

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

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

sc.AutoLoop = 1;
sc.GraphRegion = 0;

return;
}

if (!Input_Enabled.GetYesNo())
return;

SCFloatArrayRef ClosePriceArray = sc.Close;

// Use persistent variables to remember attached order IDs so they can be modified or canceled.
int& Target1OrderID = sc.GetPersistentInt(1);
int& Stop1OrderID = sc.GetPersistentInt(2);

// Calculate the moving average
sc.SimpleMovAvg(ClosePriceArray, Subgraph_SimpMovAvg, sc.Index, 10);

if (sc.IsFullRecalculation)
return;

// Create an s_SCNewOrder object.
s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 1;
NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;

NewOrder.OCOGroup1Quantity = 1; // If this is left at the default of 0, then it will be automatically set.

// Buy when the last price crosses the moving average from below.
if (sc.CrossOver(ClosePriceArray, Subgraph_SimpMovAvg) == CROSS_FROM_BOTTOM &&
sc.GetBarHasClosedStatus() == BHCS_BAR_HAS_CLOSED)
{
    NewOrder.Target1Price =sc.Close[sc.Index] + 8*sc.TickSize ;
    NewOrder.Stop1Price = sc.Close[sc.Index]- 8*sc.TickSize;

    int Result = static_cast<int>(sc.BuyEntry(NewOrder));
    if (Result > 0) //If there has been a successful order entry, then draw an arrow at the low of the bar.
    {
        Subgraph_BuyEntry[sc.Index] = sc.Low[sc.Index];

        // Remember the order IDs for subsequent modification and cancellation
        Target1OrderID = NewOrder.Target1InternalOrderID;
        Stop1OrderID = NewOrder.Stop1InternalOrderID;
    }
}

```

```

}

// Sell when the last price crosses the moving average from above.
else if (sc.CrossOver(ClosePriceArray, Subgraph_SimpMovAvg) == CROSS_FROM_TOP &&
sc.GetBarHasClosedStatus() == BHCS_BAR_HAS_CLOSED)
{
    NewOrder.Target1Price =sc.Close[sc.Index] - 8*sc.TickSize;
    NewOrder.Stop1Price =sc.Close[sc.Index] + 8*sc.TickSize;

    int Result = static_cast<int>(sc.SellEntry(NewOrder));
    if (Result > 0) //If there has been a successful order entry, then draw an arrow at the high of the bar.
    {
        Subgraph_SellEntry[sc.Index] = sc.High[sc.Index];

        // Remember the order IDs for subsequent modification and cancellation
        Target1OrderID = NewOrder.Target1InternalOrderID;
        Stop1OrderID = NewOrder.Stop1InternalOrderID;
    }
}

// This code block serves as an example of how to modify an attached order. In this example it is set to false and never
runs.
bool ExecuteModifyOrder = false;
if (ExecuteModifyOrder && (sc.Index == sc.ArraySize - 1))//Only do a modify on the most recent bar
{
    double NewLimit = 0.0;

    // Get the existing target order
    s_SCTradeOrder ExistingOrder;
    if (sc.GetOrderByOrderID(Target1OrderID, ExistingOrder) != SCTRADING_ORDER_ERROR)
    {
        if (ExistingOrder.BuySell == BSE_BUY)
            NewLimit = sc.Close[sc.Index] - 5*sc.TickSize;
        else if (ExistingOrder.BuySell == BSE_SELL)
            NewLimit = sc.Close[sc.Index] + 5*sc.TickSize;

        // We can only modify price and/or quantity

        s_SCNewOrder ModifyOrder;
        ModifyOrder.InternalOrderID = Target1OrderID;
        ModifyOrder.Price1 = NewLimit;

        sc.ModifyOrder(ModifyOrder);
    }
}

/*=====*/
SCSFExport scsf_TradingExampleWithStopAllAttachedOrdersDirectlyDefined(SCStudyInterfaceRef sc)
{
    // Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SimpMovAvg = sc.Subgraph[4];
    SCInputRef Input_Enabled = sc.Input[0];

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

        sc.GraphName = "Trading Example: With Hardcoded Attached Orders (uses Stop All)";
    }
}

```



```

Subgraph_BuyEntry.Name = "Buy Entry";
Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
Subgraph_BuyEntry.LineWidth = 2;
Subgraph_BuyEntry.DrawZeros = false;

```

```

Subgraph_SellEntry.Name = "Sell Entry";
Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
Subgraph_SellEntry.LineWidth = 2;
Subgraph_SellEntry.DrawZeros = false;

```

```

Subgraph_SimpMovAvg.Name = "Simple Moving Average";
Subgraph_SimpMovAvg.DrawStyle = DRAWSTYLE_LINE;
Subgraph_SimpMovAvg.DrawZeros = false;

```

```

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

```

sc.StudyDescription = "This study function is an example of how to use the ACSIL Trading Functions. This example uses Attached Orders defined directly within this function. This function will display a simple moving average and perform a Buy Entry when the Last price crosses the moving average from below and a Sell Entry when the Last price crosses the moving average from above. A new entry cannot occur until the Target or Stop has been hit. When an order is sent, a corresponding arrow will appear on the chart to show that an order was sent. This study will do nothing until the Enabled Input is set to Yes.";

```

sc.AllowMultipleEntriesInSameDirection = false;
sc.MaximumPositionAllowed = 5;
sc.SupportReversals = false;

```

```

// This is false by default. Orders will go to the simulation system always.
sc.SendOrdersToTradeService = false;

```

```

sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;

```

sc.SupportAttachedOrdersForTrading = false; // This can be false in this function because we specify Attached Orders directly with the order which causes this to be considered true when submitting an order.

```

sc.CancelAllOrdersOnEntriesAndReversals= true;
sc.AllowEntryWithWorkingOrders = false;
sc.CancelAllWorkingOrdersOnExit = true;

```

```

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

```

```

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

```

```

sc.AutoLoop = 1;
sc.GraphRegion = 0;

```

```

return;
}

```

```

if (!Input_Enabled.GetYesNo())
return;

```

```

SCFloatArrayRef Last = sc.Close;

```

```

// Use persistent variables to remember attached order IDs so they can be modified or canceled.
int& Target1OrderID = sc.GetPersistentInt(0);

```

```

int& Target2OrderID = sc.GetPersistentInt(1);
int& StopAllOrderID = sc.GetPersistentInt(2);

int & ExecuteModifyOrder = sc.GetPersistentInt(3);

// Calculate the moving average
sc.SimpleMovAvg>Last, Subgraph_SimpMovAvg, sc.Index, 10);

if (sc.IsFullRecalculation)
    return;

// Create an s_SCNewOrder object.
s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 2;
NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
NewOrder.TimelnForce = SCT_TIF_GOOD_TILL_CANCELED;

//Specify Targets and Stops
NewOrder.Target1Offset = 8*sc.TickSize;
NewOrder.Target2Offset = 12*sc.TickSize;
NewOrder.StopAllOffset = 5*sc.TickSize;

// Buy when the last price crosses the moving average from below.
if (sc.CrossOver>Last, Subgraph_SimpMovAvg) == CROSS_FROM_BOTTOM && sc.GetBarHasClosedStatus() ==
BHCS_BAR_HAS_CLOSED)
{
    //NewOrder.Price1 = Last[sc.Index]+ 2*sc.TickSize;
    int Result = static_cast<int>(sc.BuyEntry(NewOrder));
    if (Result > 0) //If there has been a successful order entry, then draw an arrow at the low of the bar.
    {
        Subgraph_BuyEntry[sc.Index] = sc.Low[sc.Index];
        ExecuteModifyOrder = 1;

        // Remember the order IDs for subsequent modification and cancellation
        Target1OrderID = NewOrder.Target1InternalOrderID;
        Target2OrderID = NewOrder.Target2InternalOrderID;
        StopAllOrderID = NewOrder.StopAllInternalOrderID;
    }
}

// Sell when the last price crosses the moving average from above.
else if (sc.CrossOver>Last, Subgraph_SimpMovAvg) == CROSS_FROM_TOP && sc.GetBarHasClosedStatus() ==
BHCS_BAR_HAS_CLOSED)
{
    //NewOrder.Price1 = Last[sc.Index] - 2*sc.TickSize;
    int Result = static_cast<int>(sc.SellEntry(NewOrder));
    if (Result > 0) //If there has been a successful order entry, then draw an arrow at the high of the bar.
    {
        Subgraph_SellEntry[sc.Index] = sc.High[sc.Index];
        ExecuteModifyOrder = 1;

        // Remember the order IDs for subsequent modification and cancellation
        Target1OrderID = NewOrder.Target1InternalOrderID;
        Target2OrderID = NewOrder.Target2InternalOrderID;
        StopAllOrderID = NewOrder.StopAllInternalOrderID;
    }
}

// This code block serves as an example of how to modify an Attached Order.
if (ExecuteModifyOrder == 1 && (sc.Index == sc.ArraySize - 1))//Only do a modify on the most recent bar
{

```

```

ExecuteModifyOrder = 0;

double NewPrice = 0.0;

// Get the existing target order
s_SCTradeOrder ExistingOrder;
if (sc.GetOrderByOrderID(StopAllOrderID, ExistingOrder) != SCTRADING_ORDER_ERROR)
{
    if (IsWorkingOrderStatus(ExistingOrder.OrderStatusCode))
    {
        if (ExistingOrder.BuySell == BSE_BUY)
            NewPrice = ExistingOrder.Price1 + 10*sc.TickSize;
        else if (ExistingOrder.BuySell == BSE_SELL)
            NewPrice = ExistingOrder.Price1 - 10*sc.TickSize;

        // We can only modify price and/or quantity

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

        sc.ModifyOrder(ModifyOrder);
    }
}
}

/*=====*/

```

```

SCSFExport scsf_ACSILTradingOCOExample(SCStudyInterfaceRef sc)
{
    //Define references to the Subgraphs and Inputs for easy reference

    SCInputRef Input_Enabled = sc.Input[0];

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

        sc.GraphName = "Trading OCO Order Example";

        Input_Enabled.Name = "Enabled";
        Input_Enabled.SetYesNo(0);
        Input_Enabled. SetDescription("This input enables the study and allows it to function. Otherwise, it does nothing.");

        // This is false by default. Orders will go to the simulation system always.
        sc.SendOrdersToTradeService = false;
        sc.AllowMultipleEntriesInSameDirection = false;
        sc.MaximumPositionAllowed = 10;
        sc.SupportReversals = false;
        sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;
        sc.SupportAttachedOrdersForTrading = false;
        sc.CancelAllOrdersOnEntriesAndReversals= false;
        sc.AllowEntryWithWorkingOrders = false;
        sc.CancelAllWorkingOrdersOnExit = true;
        sc.AllowOnlyOneTradePerBar = false;

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

        sc.AutoLoop = false;
        sc.GraphRegion = 0;
    }
}

```

```

    return;
}

//These must be outside of the sc.SetDefaults code block
sc.SupportTradingScaleIn = 0;
sc.SupportTradingScaleOut = 0;

if (!Input_Enabled.GetYesNo())
    return;

// Do not run on a full calculation.
if (sc.IsFullRecalculation)
    return;

float Last = sc.BaseDataIn[SC_LAST][sc.Index];

s_SCPositionData PositionData;
sc.GetTradePosition(PositionData);

if(PositionData.PositionQuantity != 0 || PositionData.WorkingOrdersExist)
    return;

s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 4;
NewOrder.OrderType = SCT_ORDERTYPE_OCO_BUY_STOP_SELL_STOP;
NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;

NewOrder.Price1 = Last + 10 * sc.TickSize;
NewOrder.Price2 = Last - 10 * sc.TickSize;

// Optional: Add target and stop Attached Orders to each stop in the OCO pair.
NewOrder.AttachedOrderTarget1Type = SCT_ORDERTYPE_LIMIT;
NewOrder.Target1Offset = 4 * sc.TickSize;
NewOrder.AttachedOrderStop1Type = SCT_ORDERTYPE_STOP;
NewOrder.Stop1Offset = 4 * sc.TickSize;

// Optional: Use different Attached Orders for the 2nd order in the OCO pair.

NewOrder.Target1Offset_2 = 8 * sc.TickSize;
//NewOrder.Stop1Offset_2 = 8 * sc.TickSize;

NewOrder.Stop1Price_2 = NewOrder.Price2 + 2;

if (sc.SubmitOCOOrder(NewOrder, sc.ArraySize - 1) > 0)
{
    int BuyStopOrder1ID = NewOrder.InternalOrderID;
    int SellStopOrder2ID = NewOrder.InternalOrderID2;
}
}

/*=====*/
SCSFExport scsf_TradingExample1WithAdvancedAttachedOrders(SCStudyInterfaceRef sc)
{
    // Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SimpMovAvg = sc.Subgraph[4];

    SCInputRef Input_Enabled = sc.Input[0];

    if (sc.SetDefaults)
    {

```

```
// Set the study configuration and defaults.
```

```
sc.GraphName = "Trading Example: 1 With Advanced Attached Orders";
```

```
Subgraph_BuyEntry.Name = "Buy Entry";  
Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;  
Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);  
Subgraph_BuyEntry.LineWidth = 2;  
Subgraph_BuyEntry.DrawZeros = false;
```

```
Subgraph_SellEntry.Name = "Sell Entry";  
Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;  
Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);  
Subgraph_SellEntry.LineWidth = 2;  
Subgraph_SellEntry.DrawZeros = false;
```

```
Subgraph_SimpMovAvg.Name = "Simple Moving Average";  
Subgraph_SimpMovAvg.DrawStyle = DRAWSTYLE_LINE;  
Subgraph_SimpMovAvg.PrimaryColor = RGB(255,255,0);  
Subgraph_SimpMovAvg.LineWidth = 2;  
Subgraph_SimpMovAvg.DrawZeros = false;
```

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

sc.StudyDescription = "This study function is an example of how to use the ACSIL Trading functions. This example uses Attached Orders defined directly within this function. It demonstrates more advanced use of Attached Orders. A new entry cannot occur until the Target or Stop has been filled or canceled. When an order is sent, a corresponding arrow will appear on the chart to show that an order was sent. This study will do nothing until the 'Enabled' Input is set to Yes.";

```
sc.AllowMultipleEntriesInSameDirection = false;  
sc.MaximumPositionAllowed = 5;  
sc.SupportReversals = false;
```

```
// This is false by default. Orders will go to the simulation system always.  
sc.SendOrdersToTradeService = false;
```

```
sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;
```

// This can be false in this function because we specify Attached Orders directly with the order which causes this to be considered true when submitting an order.

```
sc.SupportAttachedOrdersForTrading = false;
```

```
sc.CancelAllOrdersOnEntriesAndReversals= true;  
sc.AllowEntryWithWorkingOrders = false;  
sc.CancelAllWorkingOrdersOnExit = true;
```

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

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

```
sc.AutoLoop = 1;  
sc.GraphRegion = 0;
```

```
return;
```

```
}
```

```
if (!Input_Enabled.GetYesNo())  
return;
```

```
SCFloatArrayRef Last = sc.Close;
```

```

// Calculate the moving average
sc.SimpleMovAvg>Last, Subgraph_SimpMovAvg, sc.Index, 10);

// Create an s_SCNewOrder object.
s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 2;
NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
NewOrder.TimelnForce = SCT_TIF_GOOD_TILL_CANCELED;

//Define the Attached Orders to be attached to the main Market order
//Target 1
NewOrder.Target1Offset = 4 * sc.TickSize;
NewOrder.AttachedOrderTarget1Type = SCT_ORDERTYPE_LIMIT;

//Target 2
NewOrder.Target2Offset = 8 * sc.TickSize;
NewOrder.AttachedOrderTarget2Type = SCT_ORDERTYPE_LIMIT;

// Common Step Trailing Stop
NewOrder.StopAllOffset = 8*sc.TickSize;
NewOrder.AttachedOrderStopAllType = SCT_ORDERTYPE_STEP_TRAILING_STOP_LIMIT;
NewOrder.TrailStopStepPriceAmount = 4 * sc.TickSize;

// Common Trailing Stop. Comment the section above and uncomment this section to use a simple trailing stop.
//NewOrder.StopAllOffset = 8*sc.TickSize;
//NewOrder.AttachedOrderStopAllType = SCT_ORDERTYPE_TRAILING_STOP;

// Buy when the last price crosses the moving average from below.
if (sc.CrossOver>Last, Subgraph_SimpMovAvg) == CROSS_FROM_BOTTOM && sc.GetBarHasClosedStatus() ==
BHCS_BAR_HAS_CLOSED)
{
    int Result = static_cast<int>(sc.BuyEntry(NewOrder));
    if (Result > 0) //If there has been a successful order entry, then draw an arrow at the low of the bar.
    {
        Subgraph_BuyEntry[sc.Index] = sc.Low[sc.Index];
    }
}

// Sell when the last price crosses the moving average from above.
else if (sc.CrossOver>Last, Subgraph_SimpMovAvg) == CROSS_FROM_TOP && sc.GetBarHasClosedStatus() ==
BHCS_BAR_HAS_CLOSED)
{
    int Result = static_cast<int>(sc.SellEntry(NewOrder));
    if (Result > 0) //If there has been a successful order entry, then draw an arrow at the high of the bar.
    {
        Subgraph_SellEntry[sc.Index] = sc.High[sc.Index];
    }
}
}

/*=====*/
SCSFExport scsf_TradingExample2WithAdvancedAttachedOrders(SCStudyInterfaceRef sc)
{
    // Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SimpMovAvg = sc.Subgraph[4];

    SCInputRef Input_Enabled = sc.Input[0];

```

```

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

    sc.GraphName = "Trading Example: 2 With Advanced Attached Orders";

    Subgraph_BuyEntry.Name = "Buy Entry";
    Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
    Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
    Subgraph_BuyEntry.LineWidth = 2;
    Subgraph_BuyEntry.DrawZeros = false;

    Subgraph_SellEntry.Name = "Sell Entry";
    Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
    Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
    Subgraph_SellEntry.LineWidth = 2;
    Subgraph_SellEntry.DrawZeros = false;

    Subgraph_SimpMovAvg.Name = "Simple Moving Average";
    Subgraph_SimpMovAvg.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_SimpMovAvg.PrimaryColor = RGB(255,255,0);
    Subgraph_SimpMovAvg.LineWidth = 2;
    Subgraph_SimpMovAvg.DrawZeros = false;

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

    sc.StudyDescription = "This study function is an example of how to use the ACSIL Trading functions. This example
uses Attached Orders defined directly within this function. It demonstrates more advanced use of Attached Orders. A new
entry cannot occur until the Targets and Stops have been filled or canceled. When an order is sent, a corresponding
arrow will appear on the chart to show that an order was sent. This study will do nothing until the Enabled Input is set to
Yes.";

    sc.AllowMultipleEntriesInSameDirection = false;
    sc.MaximumPositionAllowed = 5;
    sc.SupportReversals = false;

    // This is false by default. Orders will go to the simulation system always.
    sc.SendOrdersToTradeService = false;

    sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;

    // This can be false in this function because we specify Attached Orders directly with the order which causes this to
be considered true when submitting an order.
    sc.SupportAttachedOrdersForTrading = false;

    sc.CancelAllOrdersOnEntriesAndReversals= true;
    sc.AllowEntryWithWorkingOrders = false;
    sc.CancelAllWorkingOrdersOnExit = true;

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

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

    sc.AutoLoop = 1;
    sc.GraphRegion = 0;

    return;
}

if (!Input_Enabled.GetYesNo())
    return;

```

```

SCFloatArrayRef Last = sc.Close;

// Calculate the moving average
sc.SimpleMovAvg(Last, Subgraph_SimpMovAvg, sc.Index, 10);

// Create an s_SCNewOrder object.
s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 2;
NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
//NewOrder.Price1; // This needs to be set if using a limit or stop order type.

//Define the Attached Orders to be attached to the main Market order
//Target 1
NewOrder.Target1Offset = 4 * sc.TickSize;
NewOrder.AttachedOrderTarget1Type = SCT_ORDERTYPE_LIMIT;

//Target 2
NewOrder.Target2Offset = 12 * sc.TickSize;
NewOrder.AttachedOrderTarget2Type = SCT_ORDERTYPE_LIMIT;

//Common Stop
NewOrder.StopAllOffset = 8 * sc.TickSize;
NewOrder.AttachedOrderStopAllType = SCT_ORDERTYPE_TRIGGERED_TRAILING_STOP_LIMIT_3_OFFSETS;
NewOrder.TriggeredTrailStopTriggerPriceOffset = 8 * sc.TickSize;
NewOrder.TriggeredTrailStopTrailPriceOffset = 4 * sc.TickSize;

//Set up a move to breakeven action for the common stop. When Target 1 is filled, the order will be moved to
breakeven +1
NewOrder.MoveToBreakEven.Type = MOVETO_BE_ACTION_TYPE_OCO_GROUP_TRIGGERED;
NewOrder.MoveToBreakEven.BreakEvenLevelOffsetInTicks = 1;
NewOrder.MoveToBreakEven.TriggerOCOGroup= OCO_GROUP_1;

// Buy when the last price crosses the moving average from below.
if (sc.Crossover(Last, Subgraph_SimpMovAvg) == CROSS_FROM_BOTTOM && sc.GetBarHasClosedStatus() ==
BHCS_BAR_HAS_CLOSED)
{
    int Result = static_cast<int>(sc.BuyEntry(NewOrder));
    if (Result > 0) //If there has been a successful order entry, then draw an arrow at the low of the bar.
    {
        Subgraph_BuyEntry[sc.Index] = sc.Low[sc.Index];
    }
}

// Sell when the last price crosses the moving average from above.
else if (sc.Crossover(Last, Subgraph_SimpMovAvg) == CROSS_FROM_TOP && sc.GetBarHasClosedStatus() ==
BHCS_BAR_HAS_CLOSED)
{
    int Result = static_cast<int>(sc.SellEntry(NewOrder));
    if (Result > 0) //If there has been a successful order entry, then draw an arrow at the high of the bar.
    {
        Subgraph_SellEntry[sc.Index] = sc.High[sc.Index];
    }
}
}
}

```

```

/*=====*/

```



```

SCSFExport scsf_TradingSystemStudySubgraphCrossover(SCStudyInterfaceRef sc)
{
    SCInputRef Input_Line1Ref = sc.Input[0];
    SCInputRef Input_Line2Ref = sc.Input[1];
    SCInputRef Input_Enabled = sc.Input[2];
    SCInputRef Input_SendTradeOrdersToTradeService = sc.Input[3];
    SCInputRef Input_MaximumPositionAllowed = sc.Input[4];
    SCInputRef Input_ActionOnCrossoverWhenInPosition = sc.Input[5];
    SCInputRef Input_EvaluateOnBarCloseOnly = sc.Input[6];

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

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

        sc.GraphName = "Trading System - Study Subgraph Crossover";

        sc.StudyDescription = "A trading system that enters a new position or \
            exits an existing one on the crossover of two study Subgraphs. \
            When Line1 crosses Line2 from below, the system will go Long. \
            When Line1 crosses Line2 from above, the system will go Short.";

        sc.AutoLoop = 1; // true
        sc.GraphRegion = 0;
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        Input_Line1Ref.Name = "Line1";
        Input_Line1Ref.SetStudySubgraphValues(1, 0);

        Input_Line2Ref.Name = "Line2";
        Input_Line2Ref.SetStudySubgraphValues(1, 0);

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

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

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

        Input_ActionOnCrossoverWhenInPosition.Name = "Action on Crossover When in Position";
        Input_ActionOnCrossoverWhenInPosition.SetCustomInputStrings("No Action;Exit on Crossover;Reverse on \
Crossover");
        Input_ActionOnCrossoverWhenInPosition.SetCustomInputIndex(0);

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

        Subgraph_BuyEntry.Name = "Buy";
        Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_POINT_ON_HIGH;
        Subgraph_BuyEntry.LineWidth = 4;

        Subgraph_SellEntry.Name = "Sell";
        Subgraph_SellEntry.DrawStyle = DRAWSTYLE_POINT_ON_LOW;
        Subgraph_SellEntry.LineWidth = 4;

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

        sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;
    }
}

```

```

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

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

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

return;
}

sc.MaximumPositionAllowed = Input_MaximumPositionAllowed.GetInt();
sc.SendOrdersToTradeService = Input_SendTradeOrdersToTradeService.GetYesNo();

if (!Input_Enabled.GetYesNo())
return;

// only process at the close of the bar. If it has not closed do not do anything
if (Input_EvaluateOnBarCloseOnly.GetYesNo()
    && sc.GetBarHasClosedStatus() == BHCS_BAR_HAS_NOT_CLOSED)
{
return;
}

// using the Input_Line1Ref and Input_Line2Ref input variables, retrieve the subgraph arrays
SCFloatArray StudyLine1;
sc.GetStudyArrayUsingID(Input_Line1Ref.GetStudyID(), Input_Line1Ref.GetSubgraphIndex(), StudyLine1);

SCFloatArray StudyLine2;
sc.GetStudyArrayUsingID(Input_Line2Ref.GetStudyID(), Input_Line2Ref.GetSubgraphIndex(), StudyLine2);

s_SCPositionData PositionData;

if (!sc.IsFullRecalculation)
    sc.GetTradePosition(PositionData);

// code below is where we check for crossovers and take action accordingly

sc.SupportReversals = false;

if (sc.CrossOver(StudyLine1, StudyLine2) == CROSS_FROM_BOTTOM)
{
    // mark the crossover on the chart
    Subgraph_BuyEntry[sc.Index] = 1;

    if (!sc.IsFullRecalculation && !sc.DownloadingHistoricalData)
    {
        if (Input_ActionOnCrossoverWhenInPosition.GetIndex() == 1
            && PositionData.PositionQuantity < 0)
        {
            s_SCNewOrder NewOrder;
            NewOrder.OrderQuantity = 0;
            NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
            NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
            int Result = static_cast<int>(sc.SellExit(NewOrder)); //Buy order
            if (Result < 0)
                sc.AddMessageToLog(sc.GetTradingErrorMessage(Result), false);
        }
        else if (Input_ActionOnCrossoverWhenInPosition.GetIndex() == 2
            && PositionData.PositionQuantity < 0)
    }
}

```

```

{
    sc.SupportReversals = true;
    s_SCNewOrder NewOrder;
    NewOrder.OrderQuantity = sc.TradeWindowOrderQuantity;
    NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
    NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
    int Result = static_cast<int>(sc.BuyEntry(NewOrder)); //Buy order
    if (Result < 0)
        sc.AddMessageToLog(sc.GetTradingErrorMessage(Result), false);
}
else if (PositionData.PositionQuantity == 0)
{
    s_SCNewOrder NewOrder;
    NewOrder.OrderQuantity = sc.TradeWindowOrderQuantity;
    NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
    NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
    int Result = static_cast<int>(sc.BuyEntry(NewOrder)); //Buy order
    if (Result < 0)
        sc.AddMessageToLog(sc.GetTradingErrorMessage(Result), false);
}
}
}

if (sc.CrossOver(StudyLine1, StudyLine2) == CROSS_FROM_TOP)
{
    // mark the crossover on the chart
    Subgraph_SellEntry[sc.Index] = 1;

    if (!sc.IsFullRecalculation && !sc.DownloadingHistoricalData)
    {
        if (Input_ActionOnCrossoverWhenInPosition.GetIndex() == 1
            && PositionData.PositionQuantity > 0)
        {
            s_SCNewOrder NewOrder;
            NewOrder.OrderQuantity = 0;
            NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
            NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
            int Result = static_cast<int>(sc.BuyExit(NewOrder)); //Sell order
            if (Result < 0)
                sc.AddMessageToLog(sc.GetTradingErrorMessage(Result), false);
        }
        else if (Input_ActionOnCrossoverWhenInPosition.GetIndex() == 2
            && PositionData.PositionQuantity > 0)
        {
            sc.SupportReversals = true;
            s_SCNewOrder NewOrder;
            NewOrder.OrderQuantity = sc.TradeWindowOrderQuantity;
            NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
            NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
            int Result = static_cast<int>(sc.SellEntry(NewOrder)); //Sell order
            if (Result < 0)
                sc.AddMessageToLog(sc.GetTradingErrorMessage(Result), false);
        }
    }
    else if (PositionData.PositionQuantity == 0)
    {
        s_SCNewOrder NewOrder;
        NewOrder.OrderQuantity = sc.TradeWindowOrderQuantity;
        NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
        NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
        int Result = static_cast<int>(sc.SellEntry(NewOrder)); //Sell order
        if (Result < 0)
            sc.AddMessageToLog(sc.GetTradingErrorMessage(Result), false);
    }
}
}

```

```
}  
}
```

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

```
SCSFExport scsf_TradingOrderEntryExamples(SCStudyInterfaceRef sc)
```

```
{  
    // Subgraph references  
    // None
```

```
    // Input references  
    SCInputRef Input_Enabled = sc.Input[0];
```

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

```
        sc.GraphName = "Trading Order Entry Examples";
```

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

```
        sc.StudyDescription = "This study function is an example of submitting various types of orders. It does not actually  
submit any orders, it only contains code examples for submitting orders.";
```

```
        sc.AllowMultipleEntriesInSameDirection = false;  
        sc.MaximumPositionAllowed = 10;//Quantity 10  
        sc.SupportReversals = false;
```

```
        // This is false by default. Orders will go to the simulation system always.  
        sc.SendOrdersToTradeService = false;
```

```
        sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;  
        sc.SupportAttachedOrdersForTrading = false;  
        sc.CancelAllOrdersOnEntriesAndReversals= true;  
        sc.AllowEntryWithWorkingOrders = false;  
        sc.CancelAllWorkingOrdersOnExit = false;
```

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

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

```
        sc.AutoLoop = 1;  
        sc.GraphRegion = 0;
```

```
        return;  
    }
```

```
    if (!Input_Enabled.GetYesNo())  
        return;
```

```
    if (0)  
    {  
        //Order entry examples  
        {
```

```
            //Move to breakeven on Stop Attached order
```

```
            // Create an s_SCNewOrder object.  
            s_SCNewOrder NewOrder;
```

```

NewOrder.OrderQuantity = 2;
NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;

//Define the Attached Orders to be attached to the main Market order
//Target 1
NewOrder.Target1Offset = 10*sc.TickSize;
NewOrder.AttachedOrderTarget1Type = SCT_ORDERTYPE_LIMIT;

//Common Stop
NewOrder.StopAllOffset = 10*sc.TickSize;
NewOrder.AttachedOrderStopAllType = SCT_ORDERTYPE_STOP;

// Set up a move to breakeven action for the common stop.

// This is a common setting and applies to all Stop Attached Orders set on the main order.
NewOrder.MoveToBreakEven.Type=MOVETO_BE_ACTION_TYPE_OFFSET_TRIGGERED;

//After 5 ticks of profit, the stop order will be moved to breakeven
NewOrder.MoveToBreakEven.TriggerOffsetInTicks= 5;
NewOrder.MoveToBreakEven.BreakEvenLevelOffsetInTicks= 0;

sc.BuyOrder(NewOrder);
}

{
// Trailing stop order. This has no Attached Orders.
// The trailing offset is the difference between the current market price for the Symbol of the chart the trading
study is applied to
// and the price set by NewOrder.Price1

// Create an s_SCNewOrder object.
s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 2;
NewOrder.OrderType = SCT_ORDERTYPE_TRAILING_STOP;
NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
NewOrder.Price1 = sc.BaseData[SC_LAST][sc.Index] - 10*sc.TickSize;
sc.SellOrder(NewOrder);
}

{
// Trailing Stop Attached Order
// The trailing offset is set by NewOrder.Stop1Offset

// Create an s_SCNewOrder object.
s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 2;
NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
NewOrder.AttachedOrderStop1Type = SCT_ORDERTYPE_TRAILING_STOP;
NewOrder.Stop1Offset = 10 * sc.TickSize;
sc.BuyOrder(NewOrder);
}
}

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

```

```

// Define references to the Subgraphs and Inputs for easy reference
SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[1];
SCSubgraphRef Subgraph_FastSimpMovAvg = sc.Subgraph[4];
SCSubgraphRef Subgraph_SlowSimpMovAvg = sc.Subgraph[5];

SCInputRef Input_Enabled = sc.Input[0];

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

    sc.GraphName = "Trading Example: Moving Average Crossover with Attached Orders";

    Subgraph_BuyEntry.Name = "Buy Entry";
    Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
    Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
    Subgraph_BuyEntry.LineWidth = 2;
    Subgraph_BuyEntry.DrawZeros = false;

    Subgraph_SellEntry.Name = "Sell Entry";
    Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
    Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
    Subgraph_SellEntry.LineWidth = 2;
    Subgraph_SellEntry.DrawZeros = false;

    Subgraph_FastSimpMovAvg.Name = "Fast Moving Average";
    Subgraph_FastSimpMovAvg.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_FastSimpMovAvg.PrimaryColor = RGB(255, 255, 255);
    Subgraph_FastSimpMovAvg.DrawZeros = false;
    Subgraph_FastSimpMovAvg.LineWidth = 2;

    Subgraph_SlowSimpMovAvg.Name = "Slow Moving Average";
    Subgraph_SlowSimpMovAvg.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_SlowSimpMovAvg.PrimaryColor = RGB(0, 255, 0);
    Subgraph_SlowSimpMovAvg.DrawZeros = false;
    Subgraph_SlowSimpMovAvg.LineWidth = 2;

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

    sc.AllowMultipleEntriesInSameDirection = false;
    sc.MaximumPositionAllowed = 2;
    sc.SupportReversals = false;

    // This is false by default. Orders will go to the simulation system always.
    sc.SendOrdersToTradeService = false;

    sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;

    // This can be false in this function because we specify Attached Orders directly with the order which causes this to
    // be considered true when submitting an order.
    sc.SupportAttachedOrdersForTrading = false;

    sc.CancelAllOrdersOnEntriesAndReversals = false;
    sc.AllowEntryWithWorkingOrders = false;
    sc.CancelAllWorkingOrdersOnExit = true;

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

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

```

```

sc.AutoLoop = 1;
sc.GraphRegion = 0;

return;
}

if (!Input_Enabled.GetYesNo())
return;

// Use persistent variables to remember attached order IDs so they can be modified or canceled.
int& Target1OrderID = sc.GetPersistentInt(1);
int& Stop1OrderID = sc.GetPersistentInt(2);

// Calculate the moving average
sc.SimpleMovAvg(sc.Close, Subgraph_FastSimpMovAvg, 10);
sc.SimpleMovAvg(sc.Close, Subgraph_SlowSimpMovAvg, 20);

s_SCPositionData PositionData;
sc.GetTradePosition(PositionData);
if(PositionData.PositionQuantity != 0)
return;

// Create an s_SCNewOrder object.
s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 1;
NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
NewOrder.TimelnForce = SCT_TIF_GOOD_TILL_CANCELED;

//Specify a Target and a Stop with 8 tick offsets. We are specifying one Target and one Stop, additional targets and
stops can be specified as well.
NewOrder.Target1Offset = 8*sc.TickSize;
NewOrder.Stop1Offset = 8*sc.TickSize;
NewOrder.OCOGroup1Quantity = 1; // If this is left at the default of 0, then it will be automatically set.

// Buy when the last price crosses the moving average from below.
if (sc.CrossOver(Subgraph_FastSimpMovAvg, Subgraph_SlowSimpMovAvg) == CROSS_FROM_BOTTOM &&
sc.GetBarHasClosedStatus() == BHCS_BAR_HAS_CLOSED)
{
int Result = static_cast<int>(sc.BuyEntry(NewOrder));
if (Result > 0) //If there has been a successful order entry, then draw an arrow at the low of the bar.
{
Subgraph_BuyEntry[sc.Index] = sc.Low[sc.Index];

// Remember the order IDs for subsequent modification and cancellation
Target1OrderID = NewOrder.Target1InternalOrderID;
Stop1OrderID = NewOrder.Stop1InternalOrderID;
}
}

// Sell when the last price crosses the moving average from above.
else if (sc.CrossOver(Subgraph_FastSimpMovAvg, Subgraph_SlowSimpMovAvg) == CROSS_FROM_TOP &&
sc.GetBarHasClosedStatus() == BHCS_BAR_HAS_CLOSED)
{
int Result = static_cast<int>(sc.SellEntry(NewOrder));
if (Result > 0) //If there has been a successful order entry, then draw an arrow at the high of the bar.
{
Subgraph_SellEntry[sc.Index] = sc.High[sc.Index];

// Remember the order IDs for subsequent modification and cancellation
Target1OrderID = NewOrder.Target1InternalOrderID;
Stop1OrderID = NewOrder.Stop1InternalOrderID;
}
}

```

```

}

}

/*=====*/
SCSFExport scsf_TradingExampleModifyStopAttachedOrders(SCStudyInterfaceRef sc)
{
    // Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[1];
    SCSubgraphRef Subgraph_FastSimpMovAvg = sc.Subgraph[4];
    SCSubgraphRef Subgraph_SlowSimpMovAvg = sc.Subgraph[5];

    SCInputRef Input_Enabled = sc.Input[0];

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

        sc.GraphName = "Trading Example: Modify Stop Attached Orders";

        Subgraph_BuyEntry.Name = "Buy Entry";
        Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
        Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
        Subgraph_BuyEntry.LineWidth = 2;
        Subgraph_BuyEntry.DrawZeros = false;

        Subgraph_SellEntry.Name = "Sell Entry";
        Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
        Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
        Subgraph_SellEntry.LineWidth = 2;
        Subgraph_SellEntry.DrawZeros = false;

        Subgraph_FastSimpMovAvg.Name = "Fast Moving Average";
        Subgraph_FastSimpMovAvg.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_FastSimpMovAvg.PrimaryColor = RGB(255, 255, 255);
        Subgraph_FastSimpMovAvg.DrawZeros = false;
        Subgraph_FastSimpMovAvg.LineWidth = 2;

        Subgraph_SlowSimpMovAvg.Name = "Slow Moving Average";
        Subgraph_SlowSimpMovAvg.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SlowSimpMovAvg.PrimaryColor = RGB(0, 255, 0);
        Subgraph_SlowSimpMovAvg.DrawZeros = false;
        Subgraph_SlowSimpMovAvg.LineWidth = 2;

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

        sc.AllowMultipleEntriesInSameDirection = false;
        sc.MaximumPositionAllowed = 2;
        sc.SupportReversals = false;

        // This is false by default.
        sc.SendOrdersToTradeService = false;

        sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;

        sc.SupportAttachedOrdersForTrading = false;

        sc.CancelAllOrdersOnEntriesAndReversals = false;
        sc.AllowEntryWithWorkingOrders = false;
        sc.CancelAllWorkingOrdersOnExit = true;
    }
}

```



```

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

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

sc.AutoLoop = 1;
sc.GraphRegion = 0;

return;
}

if (!Input_Enabled.GetYesNo())
return;

int OrderIndex = 0;
while (true)
{
    // Get the next order details structure.
    s_SCTradeOrder OrderDetails;
    if (sc.GetOrderByIndex(OrderIndex, OrderDetails) == SCTRADING_ORDER_ERROR)
        break;

    ++OrderIndex;

    // Skip if not a working order.
    if (!OrderDetails.IsWorking())
        continue;

    // Skip if not an attached order.
    if (!OrderDetails.IsAttachedOrder())
        continue;

    // Skip if not a stop order.
    if (OrderDetails.OrderTypeAsInt != SCT_ORDERTYPE_STOP)
        continue;

    //Modify order to same price. Since the price is the same, it will be ignored by the next line of code. This code only
    serves as an example of an order modification.
    double NewPrice = OrderDetails.Price1;

    //If modifying to the same price as previously, then do not perform the modification
    if(NewPrice == OrderDetails.LastModifyPrice1)
        continue;

    //Modify order
    s_SCNewOrder ModifyOrder;

    ModifyOrder.InternalOrderID = OrderDetails.InternalOrderID;

    ModifyOrder.Price1 = NewPrice;

    sc.ModifyOrder(ModifyOrder);
}
}

/*=====*/
SCSFExport scsf_TradingExampleGetPositionData(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the study configuration and defaults.

```

```

sc.GraphName = "Trading Example: Get Position Data";

sc.AllowMultipleEntriesInSameDirection = false;
sc.MaximumPositionAllowed = 2;
sc.SupportReversals = false;

// This is false by default.
sc.SendOrdersToTradeService = false;

sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;

sc.SupportAttachedOrdersForTrading = false;

sc.CancelAllOrdersOnEntriesAndReversals= false;
sc.AllowEntryWithWorkingOrders = false;
sc.CancelAllWorkingOrdersOnExit = true;

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

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

sc.AutoLoop = 0;
sc.GraphRegion = 0;

return;
}

if (sc.GetBarHasClosedStatus(sc.UpdateStartIndex) != BHCS_BAR_HAS_CLOSED)
return;

s_SCPositionData PositionData;
sc.GetTradePosition(PositionData);

SCString PositionDataMessage;

PositionDataMessage.Format("MaximumOpenPositionProfit: %f. MaximumOpenPositionLoss: %f."
, PositionData.MaximumOpenPositionProfit
, PositionData.MaximumOpenPositionLoss);

sc.AddMessageToLog(PositionDataMessage, 0);

}

/*=====*/
SCSFExport scsf_TradingExampleIteratingOrderList(SCStudyInterfaceRef sc)
{
if (sc.SetDefaults)
{
// Set the study configuration and defaults.

sc.GraphName = "Trading Example: Iterating Order List";

// This is false by default.
sc.SendOrdersToTradeService = false;

sc.MaintainTradeStatisticsAndTradesData = true;

sc.AutoLoop = 0;
sc.GraphRegion = 0;

```

```

    return;
}

// This only serves as a code example and we do not want this code to run.
// return;

// This is an example of iterating the order list in Sierra Chart for orders
// matching the Symbol and Trade Account of the chart, and finding the orders
// that have a working Order Status and are not Attached Orders.

//The iteration must always be from zero until SCTRADING_ORDER_ERROR is returned. Do not reverse iterate
because that is highly inefficient. sc.GetOrderByIndex is less efficient than using sc.GetOrderByOrderID. So it should not
be called frequently.
int Index = 0;
s_SCTradeOrder OrderDetails;
while( sc.GetOrderByIndex(Index, OrderDetails) != SCTRADING_ORDER_ERROR)
{
    ++Index; // Increment the index for the next call to sc.GetOrderByIndex

    if (!IsWorkingOrderStatus(OrderDetails.OrderStatusCode))
        continue;

    if (OrderDetails.ParentInternalOrderID != 0) //This means this is an Attached Order
        continue;

    //Get the internal order ID
    int InternalOrderID = OrderDetails.InternalOrderID;
}

int TargetInternalOrderID = -1;
int StopInternalOrderID = -1;

//This needs to be set to the parent internal order ID search for. Since we do not know what that is in this code
example, it is set to zero here.
int ParentInternalOrderID = 0;

sc.GetAttachedOrderIDsForParentOrder(ParentInternalOrderID, TargetInternalOrderID, StopInternalOrderID);
}

/*=====*/
SCSFExport scsf_TradingExampleOrdersForDifferentSymbolAndAccount(SCStudyInterfaceRef sc)
{
    //Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BuyExit = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SellExit = sc.Subgraph[3];

    SCInputRef Input_Enabled = sc.Input[0];
    SCInputRef Input_ReferenceChartForPrice = sc.Input[1];

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

        sc.GraphName = "Trading Example: Orders for Different Symbol and Account";

        sc.StudyDescription = "This function demonstrates submitting, modifying and canceling an order for a different
symbol than the chart the trading study is applied to";
    }
}

```

```

Subgraph_BuyEntry.Name = "Buy Entry";
Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
Subgraph_BuyEntry.LineWidth = 2;
Subgraph_BuyEntry.DrawZeros = false;

Subgraph_BuyExit.Name = "Buy Exit";
Subgraph_BuyExit.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_BuyExit.PrimaryColor = RGB(255, 128, 128);
Subgraph_BuyExit.LineWidth = 2;
Subgraph_BuyExit.DrawZeros = false;

Subgraph_SellEntry.Name = "Sell Entry";
Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
Subgraph_SellEntry.LineWidth = 2;
Subgraph_SellEntry.DrawZeros = false;

Subgraph_SellExit.Name = "Sell Exit";
Subgraph_SellExit.DrawStyle = DRAWSTYLE_ARROW_UP;
Subgraph_SellExit.PrimaryColor = RGB(128, 255, 128);
Subgraph_SellExit.LineWidth = 2;
Subgraph_SellExit.DrawZeros = false;

Input_Enabled.Name = "Enabled";
Input_Enabled.SetYesNo(0);
Input_Enabled.SetDescription("This input enables the study and allows it to function. Otherwise, it does nothing.");

Input_ReferenceChartForPrice.Name = "Reference Chart For Price";
Input_ReferenceChartForPrice.SetChartNumber(0);

// This is false by default. Orders will be simulated.
sc.SendOrdersToTradeService = false;

// These variables are not used when trading another Symbol and/or Trade Account compared to what the chart is
set to.
sc.AllowMultipleEntriesInSameDirection = true;
sc.MaximumPositionAllowed = 20000;
sc.SupportReversals = true;
sc.AllowOppositeEntryWithOpposingPositionOrOrders = true;
sc.SupportAttachedOrdersForTrading = false;
sc.UseGUIAttachedOrderSetting = false;
sc.CancelAllOrdersOnEntriesAndReversals= true;
sc.AllowEntryWithWorkingOrders = true;
sc.CancelAllWorkingOrdersOnExit = true;

//
sc.AllowOnlyOneTradePerBar = false;

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

sc.ReceiveNotificationsForChangesToOrdersPositionsForAnySymbol = true;

sc.AutoLoop = true;
sc.GraphRegion = 0;

return;
}

//These must be outside of the sc.SetDefaults code block since they have a dependency upon an actual chart object
existing. They are not used in this example.
sc.SupportTradingScaleIn = 0;
sc.SupportTradingScaleOut = 0;

```

```

int& r_BarIndexOfOrder = sc.GetPersistentInt(1);
int& r_InternalOrderID = sc.GetPersistentInt(2);
int& r_PerformedOrderModification = sc.GetPersistentInt(3);

if (!Input_Enabled.GetYesNo())
{
    r_BarIndexOfOrder = 0;
    r_InternalOrderID = 0;
    r_PerformedOrderModification = 0;

    return;
}

if (sc.LastCallToFunction)
    return; //nothing to do

if (sc.ChartIsDownloadingHistoricalData(Input_ReferenceChartForPrice.GetChartNumber())
    || sc.IsFullRecalculation
    || sc.ServerConnectionState != SCS_CONNECTED
)
{
    return;
}

// Run the below code only on the last bar
if (sc.Index < sc.ArraySize - 1)
    return;

SCFloatArray LastPriceArrayFromChartToTrade;
sc.GetCharArray
( Input_ReferenceChartForPrice.GetChartNumber()
, SC_LAST
, LastPriceArrayFromChartToTrade
);

if (LastPriceArrayFromChartToTrade.GetArraySize() == 0)
    return;

int LastIndexOfArrayFromChartToTrade = LastPriceArrayFromChartToTrade.GetArraySize() - 1;

const char* SymbolToTrade = "XAUUSD";
const char* TradeAccount = "Sim3";

const int OrderQuantity = 100;

s_SCPositionData PositionData;
sc.GetTradePositionForSymbolAndAccount(PositionData, SymbolToTrade, sc.SelectedTradeAccount);
if (PositionData.PositionQuantity == 0 && PositionData.WorkingOrdersExist == 0 && r_InternalOrderID == 0)
{
    // Create an s_SCNewOrder object.
    s_SCNewOrder NewOrder;
    NewOrder.OrderQuantity = OrderQuantity;
    NewOrder.OrderType = SCT_ORDERTYPE_LIMIT;
    NewOrder.TimeInForce = SCT_TIF_DAY;
    NewOrder.Price1
        = LastPriceArrayFromChartToTrade[LastIndexOfArrayFromChartToTrade] - 10 * sc.TickSize;

    //Specify a Target and a Stop with 8 tick offsets. Only 1 Target and 1 Stop can be supported when trading a Symbol
    and Trade Account different than the chart the trading system study is applied to.
    NewOrder.Target1Offset = 8 * sc.TickSize;
    NewOrder.Stop1Offset = 8 * sc.TickSize;

    // If this is left at the default of 0, then it will be automatically set.

```

```
NewOrder.OCOGroup1Quantity = OrderQuantity;

NewOrder.Symbol = SymbolToTrade;
NewOrder.TradeAccount = TradeAccount; //sc.SelectedTradeAccount;

int Result = static_cast<int>(sc.BuyOrder(NewOrder));

if (Result > 0)
{
    r_BarIndexOfOrder = sc.Index;
    r_InternalOrderID = NewOrder.InternalOrderID;
}
}

else if (r_InternalOrderID != 0)
{
    s_SCTradeOrder ExistingOrderDetails;
    if (sc.GetOrderByOrderID(r_InternalOrderID, ExistingOrderDetails))
    {
        //If original submitted order is still working.
        if (IsWorkingOrderStatus(ExistingOrderDetails.OrderStatusCode) )
        {
            //When 1 new bar is added to chart since the original order submission
            if (r_PerformedOrderModification == 0 && sc.Index >= r_BarIndexOfOrder + 1)
            {
                //Modify the order
                s_SCNewOrder ModifyOrder;
                ModifyOrder.InternalOrderID = r_InternalOrderID;
                ModifyOrder.Price1 = LastPriceArrayFromChartToTrade[LastIndexOfArrayFromChartToTrade] - 12 *
sc.TickSize;

                if (sc.ModifyOrder(ModifyOrder) > 0)
                    r_PerformedOrderModification = 1;
            }
            //When 2 new bars added to chart since the original order submission
            else if (sc.Index >= r_BarIndexOfOrder + 2)
            {
                //Cancel the order
                sc.CancelOrder(r_InternalOrderID);
            }
        }
    }
}
}
```

```

/*=====*/
SCSFExport scsf_TradingExampleWithSingleAttachedOrder(SCStudyInterfaceRef sc)
{
    // Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];

    SCInputRef Input_Enabled = sc.Input[0];

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

        sc.GraphName = "Trading Example: With Single Attached Order";
    }
}

```

```

Subgraph_BuyEntry.Name = "Buy Entry";
Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
Subgraph_BuyEntry.LineWidth = 2;
Subgraph_BuyEntry.DrawZeros = false;

Subgraph_SellEntry.Name = "Sell Entry";
Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
Subgraph_SellEntry.LineWidth = 2;
Subgraph_SellEntry.DrawZeros = false;

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

sc.StudyDescription = "This study function is an example of how to use the ACSIL Trading Functions.\
  This example uses a single Attached Order defined directly within this function.";

sc.AllowMultipleEntriesInSameDirection = false;
sc.MaximumPositionAllowed = 5;
sc.SupportReversals = false;

// This is false by default. Orders will go to the simulation system always.
sc.SendOrdersToTradeService = false;

sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;

// This can be false in this function because we specify Attached Orders directly with the order which causes this to
// be considered true when submitting an order.
sc.SupportAttachedOrdersForTrading = false;

sc.CancelAllOrdersOnEntriesAndReversals = false;
sc.AllowEntryWithWorkingOrders = false;
sc.CancelAllWorkingOrdersOnExit = true;

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

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

sc.AutoLoop = 1;
sc.GraphRegion = 0;

return;
}

if (!Input_Enabled.GetYesNo())
return;

// Use persistent variables to remember attached order IDs so they can be modified or canceled.
int& Stop1OrderID = sc.GetPersistentInt(2);

// Create an s_SCNewOrder object.
s_SCNewOrder NewOrder;
NewOrder.OrderQuantity = 1;
NewOrder.OrderType = SCT_ORDERTYPE_LIMIT;
NewOrder.TimeInForce = SCT_TIF_DAY;

//Specify a Stop with 8 tick offsets. We are specifying one Stop.
NewOrder.Stop1Offset = 8*sc.TickSize;
NewOrder.OCOGroup1Quantity = 1; // If this is left at the default of 0, then it will be automatically set.

```

```

// This is where you can define the Buy condition
if (0)
{
    int Result = static_cast<int>(sc.BuyEntry(NewOrder));
    if (Result > 0) //If there has been a successful order entry, then draw an arrow at the low of the bar.
    {
        Subgraph_BuyEntry[sc.Index] = sc.Low[sc.Index];

        // Remember the order ID for subsequent modification and cancellation
        Stop1OrderID = NewOrder.Stop1InternalOrderID;
    }
}

else if (0) //Sell condition
{
    int Result = static_cast<int>(sc.SellEntry(NewOrder));
    if (Result > 0) //If there has been a successful order entry, then draw an arrow at the high of the bar.
    {
        Subgraph_SellEntry[sc.Index] = sc.High[sc.Index];

        // Remember the order ID for subsequent modification and cancellation
        Stop1OrderID = NewOrder.Stop1InternalOrderID;
    }
}
}

/*=====*/

SCSFExport scsf_TradingScaleInExample(SCStudyInterfaceRef sc)
{
    //Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BuyExit = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SellExit = sc.Subgraph[3];

    SCInputRef Input_Enabled = sc.Input[0];
    SCInputRef Input_TargetValue = sc.Input[1];
    SCInputRef Input_StopValue = sc.Input[2];

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

        sc.GraphName = "Trading Example: Scale In";

        Subgraph_BuyEntry.Name = "Buy Entry";
        Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
        Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
        Subgraph_BuyEntry.LineWidth = 2;
        Subgraph_BuyEntry.DrawZeros = false;

        Subgraph_BuyExit.Name = "Buy Exit";
        Subgraph_BuyExit.DrawStyle = DRAWSTYLE_ARROW_DOWN;
        Subgraph_BuyExit.PrimaryColor = RGB(255, 128, 128);
        Subgraph_BuyExit.LineWidth = 2;
        Subgraph_BuyExit.DrawZeros = false;

        Subgraph_SellEntry.Name = "Sell Entry";
        Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
        Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
        Subgraph_SellEntry.LineWidth = 2;
    }
}

```



```

Subgraph_SellEntry.DrawZeros = false;

Subgraph_SellExit.Name = "Sell Exit";
Subgraph_SellExit.DrawStyle = DRAWSTYLE_ARROW_UP;
Subgraph_SellExit.PrimaryColor = RGB(128, 255, 128);
Subgraph_SellExit.LineWidth = 2;
Subgraph_SellExit.DrawZeros = false;

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

Input_TargetValue.Name = "Target Offset in Ticks";
Input_TargetValue.SetFloat(10.0f);

Input_StopValue.Name = "Stop Offset in Ticks";
Input_StopValue.SetFloat(10.0f);

sc.StudyDescription = "This study function is an example of how to use the ACSIL Trading Functions and increase
the Trade Position quantity by scaling in.";

sc.AutoLoop = 1;
sc.GraphRegion = 0;

//Any of the following variables can also be set outside and below the sc.SetDefaults code block

sc.AllowMultipleEntriesInSameDirection = true;
sc.MaximumPositionAllowed = 2;
sc.SupportReversals = false;

// This is false by default. Orders will go to the simulation system always.
sc.SendOrdersToTradeService = false;

sc.AllowOppositeEntryWithOpposingPositionOrOrders = false;

//This must be true for Scale In to work
sc.SupportAttachedOrdersForTrading = true;

sc.CancelAllOrdersOnEntriesAndReversals= false;
sc.AllowEntryWithWorkingOrders = true;
sc.CancelAllWorkingOrdersOnExit = false;

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

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

return;
}

sc.SupportTradingScaleIn = true;
sc.SupportTradingScaleOut = false;

if (!Input_Enabled.GetYesNo())
return;

// Get the Trade Position data
s_SCPositionData PositionData;
sc.GetTradePosition(PositionData) ;

int& r_BuyEntryInternalOrderID = sc.GetPersistentInt(1);

// This Boolean needs to be set to true when the condition for your initial entry has been met. This trading example

```

does not actually define any entry condition but has code to enter a buy order when an entry condition has occurred.

```
bool ConditionForInitialBuyEntry = true;
bool ConditionForSecondBuyEntry = true;

// Buy when the last price crosses the moving average from below.
if (ConditionForInitialBuyEntry && PositionData.PositionQuantity == 0
    && !PositionData.WorkingOrdersExist)
{
    // Create an s_SCNewOrder object.
    s_SCNewOrder NewOrder;
    NewOrder.OrderQuantity = 1;
    NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
    NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
    NewOrder.Target1Offset = Input_TargetValue.GetFloat() * sc.TickSize;
    NewOrder.Stop1Offset = Input_StopValue.GetFloat() * sc.TickSize;

    int Result = static_cast<int>(sc.BuyEntry(NewOrder));
    //If there has been a successful order entry, then draw an arrow at the low of the bar.
    if (Result > 0)
    {
        r_BuyEntryInternalOrderID = NewOrder.InternalOrderID;
        Subgraph_BuyEntry[sc.Index] = sc.Low[sc.Index];
    }
}
else if (ConditionForSecondBuyEntry
    && PositionData.PositionQuantity == 1
    && !PositionData.NonAttachedWorkingOrdersExist
)
{
    // Create an s_SCNewOrder object.
    s_SCNewOrder NewOrder;
    NewOrder.OrderQuantity = 1;
    NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
    NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;

    int Result = static_cast<int>(sc.BuyEntry(NewOrder));
    //If there has been a successful order entry, then draw an arrow at the low of the bar.
    if (Result > 0)
    {
        r_BuyEntryInternalOrderID = NewOrder.InternalOrderID;
        Subgraph_BuyEntry[sc.Index] = sc.Low[sc.Index];
    }
}
}

/*=====*/
//This is used for testing only. It does not serve as an example.

SCSFExport scsf_ACSILTradingTest2(SCStudyInterfaceRef sc)
{
    //Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BuyExit = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SellExit = sc.Subgraph[3];

    SCInputRef Input_Enabled = sc.Input[0];

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

        sc.GraphName = "Trading Test 2";
```

```

Subgraph_BuyEntry.Name = "Buy Entry";
Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
Subgraph_BuyEntry.LineWidth = 2;
Subgraph_BuyEntry.DrawZeros = false;

Subgraph_BuyExit.Name = "Buy Exit";
Subgraph_BuyExit.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_BuyExit.PrimaryColor = RGB(255, 128, 128);
Subgraph_BuyExit.LineWidth = 2;
Subgraph_BuyExit.DrawZeros = false;

Subgraph_SellEntry.Name = "Sell Entry";
Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
Subgraph_SellEntry.LineWidth = 2;
Subgraph_SellEntry.DrawZeros = false;

Subgraph_SellExit.Name = "Sell Exit";
Subgraph_SellExit.DrawStyle = DRAWSTYLE_ARROW_UP;
Subgraph_SellExit.PrimaryColor = RGB(128, 255, 128);
Subgraph_SellExit.LineWidth = 2;
Subgraph_SellExit.DrawZeros = false;

Input_Enabled.Name = "Enabled";
Input_Enabled.SetYesNo(0);
Input_Enabled.SetDescription("This Input enables the study and allows it to function. Otherwise, it does nothing.");

// This is false by default. Orders will be simulated.
sc.SendOrdersToTradeService = false;

sc.AllowMultipleEntriesInSameDirection = false;

//This must be equal to or greater than the order quantities you will be submitting orders for.
sc.MaximumPositionAllowed = 4;

sc.SupportReversals = true;
sc.AllowOppositeEntryWithOpposingPositionOrOrders = true;

// This variable controls whether to use or not use attached orders that are configured on the Trade Window for the
chart.
sc.SupportAttachedOrdersForTrading = false;

//This variable controls whether to use the "Use Attached Orders" setting on the Trade Window for the chart
sc.UseGUIAttachedOrderSetting = false;

sc.CancelAllOrdersOnEntriesAndReversals = false;
sc.AllowEntryWithWorkingOrders = false;
sc.CancelAllWorkingOrdersOnExit = true;
sc.AllowOnlyOneTradePerBar = false;

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

sc.AutoLoop = false;
sc.GraphRegion = 0;

return;
}

//These must be outside of the sc.SetDefaults code block since they have a dependency upon an actual chart object
existing
sc.SupportTradingScaleIn = 0;
sc.SupportTradingScaleOut = 0;

```

```

//sc.SendOrdersToTradeService = true;

//s_SCTradeOrder TradeOrder;
//int Result = sc.GetNearestStopOrder(TradeOrder);

//if (Result)
//{
// int Test = 0;
//}

if (!Input_Enabled.GetYesNo())
    return;

if (sc.LastCallToFunction)
    return; //nothing to do

// Run the below code only on the last bar
if (sc.IsFullRecalculation)
    return;

//sc.AllowMultipleEntriesInSameDirection = true;
//sc.AllowOnlyOneTradePerBar = false;
//sc.AllowEntryWithWorkingOrders = true;
//sc.CancelAllWorkingOrdersOnExit = true;

s_SCPositionData PositionData;
sc.GetTradePosition(PositionData);

if (PositionData.PositionQuantity == 0)
{
    s_SCNewOrder NewOrder;
    NewOrder.OrderQuantity = 4;
    NewOrder.Price1 = sc.BaseDataIn[SC_LAST][sc.ArraySize - 1] - (20 * sc.TickSize);
    NewOrder.OrderType = SCT_ORDERTYPE_LIMIT;
    NewOrder.Target1Offset = 8 * sc.TickSize;
    NewOrder.Stop1Offset = 8 * sc.TickSize;
    //NewOrder.AttachedOrderStop1Type = SCT_ORDERTYPE_TRAILING_STOP;
    NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
    NewOrder.TextTag = "Tag";
    //NewOrder.Stop1Offset = 10 * sc.TickSize;
    sc.BuyEntry(NewOrder, sc.ArraySize - 1);

    //s_SCNewOrder NewOrder1;
    //NewOrder1.OrderQuantity = 1;
    //NewOrder1.OrderType = SCT_ORDERTYPE_MARKET;
    //NewOrder1.AttachedOrderStop1Type = SCT_ORDERTYPE_STOP_WITH_BID_ASK_TRIGGERING;
    //NewOrder1.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
    //NewOrder1.Stop1Offset = 15 * sc.TickSize;
    //sc.BuyEntry(NewOrder1, sc.ArraySize - 1);
}

//if (PositionData.PositionQuantity != 0)
//{
// sc.FlattenAndCancelAllOrders();
//}

//s_SCNewOrder NewOrder;
//NewOrder.OrderQuantity = 1;
//NewOrder.OrderType = SCT_ORDERTYPE_TRIGGERED_TRAILING_STOP_3_OFFSETS;
//NewOrder.Price1 = 1000; // Stop price
//NewOrder.Price2 = 5000; // Trail trigger price
//NewOrder.TimeInForce = SCT_TIF_DAY;

```

```

//int Result = (int)sc.SellEntry(NewOrder, sc.ArraySize - 1);

//s_SCNewOrder NewOrder;
//NewOrder.OrderQuantity = 1;
//NewOrder.OrderType = SCT_ORDERTYPE_LIMIT;
//NewOrder.Price1 = sc.Close[sc.ArraySize - 1] - sc.TickSize*10;// Limit price
//NewOrder.TimeInForce = SCT_TIF_DAY;
//NewOrder.AttachedOrderStop1Type = SCT_ORDERTYPE_TRIGGERED_TRAILING_STOP_3_OFFSETS;
//NewOrder.Stop1Offset = sc.TickSize * 10;
//NewOrder.TriggeredTrailStopTrailPriceOffset = sc.TickSize * 8;
//NewOrder.TriggeredTrailStopTriggerPriceOffset = sc.TickSize * 5;
//int Result = (int)sc.BuyEntry(NewOrder, sc.ArraySize - 1);

if (PositionData.WorkingOrdersExist)
    return;

//{
// s_SCNewOrder NewOrder;
// NewOrder.OrderQuantity = 2;
// //NewOrder.OrderType = SCT_ORDERTYPE_TRIGGERED_LIMIT;
// NewOrder.OrderType = SCT_ORDERTYPE_TRIGGERED_LIMIT_WITH_CHASE;
// NewOrder.MaximumChaseAsPrice = 5 * sc.TickSize;
// NewOrder.Price1 = sc.Close[sc.ArraySize - 1] + sc.TickSize * 10;// Limit price
// NewOrder.Price2 = sc.Close[sc.ArraySize - 1] + sc.TickSize * 20;// Stop price
// NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
// int Result = (int)sc.BuyEntry(NewOrder, sc.ArraySize - 1);
//}

//s_SCNewOrder NewOrder;
//NewOrder.OrderQuantity = 1;
//NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
////NewOrder.Price1 = 0;
//NewOrder.Target1Offset = 20 * sc.TickSize;
//NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
//NewOrder.AttachedOrderStop1Type = SCT_ORDERTYPE_TRIGGERED_TRAILING_STOP_3_OFFSETS;

//NewOrder.Stop1Offset = 20 * sc.TickSize;
//NewOrder.AttachedOrderStop1_TriggeredTrailStopTriggerPriceOffset = 10 * sc.TickSize;
//NewOrder.AttachedOrderStop1_TriggeredTrailStopTrailPriceOffset = 5 * sc.TickSize;
//sc.BuyEntry(NewOrder, sc.ArraySize - 1);
//
//s_SCNewOrder NewOrder;
//NewOrder.OrderQuantity = 1;
//NewOrder.OrderType = SCT_ORDERTYPE_TRIGGERED_LIMIT;
//NewOrder.Price1 = 0;//set this to the Limit price
//NewOrder.Price2 = 0;//set this to the Trigger price
//sc.BuyEntry(NewOrder, sc.ArraySize - 1);

}

/*=====*/
SCSFExport scsf_GetOrderFillEntryExample(SCStudyInterfaceRef sc)
{
    //Define references to the Subgraphs and Inputs for easy reference.
    //In this particular example, these are not used but this study is configured as a typical trading study.
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BuyExit = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SellExit = sc.Subgraph[3];

    SCInputRef Input_Enabled = sc.Input[0];

    if (sc.SetDefaults)
    {

```

```

// Set the study configuration and defaults.

sc.GraphName = "GetOrderFillEntry Example";

Subgraph_BuyEntry.Name = "Buy Entry";
Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
Subgraph_BuyEntry.LineWidth = 2;
Subgraph_BuyEntry.DrawZeros = false;

Subgraph_BuyExit.Name = "Buy Exit";
Subgraph_BuyExit.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_BuyExit.PrimaryColor = RGB(255, 128, 128);
Subgraph_BuyExit.LineWidth = 2;
Subgraph_BuyExit.DrawZeros = false;

Subgraph_SellEntry.Name = "Sell Entry";
Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
Subgraph_SellEntry.LineWidth = 2;
Subgraph_SellEntry.DrawZeros = false;

Subgraph_SellExit.Name = "Sell Exit";
Subgraph_SellExit.DrawStyle = DRAWSTYLE_ARROW_UP;
Subgraph_SellExit.PrimaryColor = RGB(128, 255, 128);
Subgraph_SellExit.LineWidth = 2;
Subgraph_SellExit.DrawZeros = false;

Input_Enabled.Name = "Enabled";
Input_Enabled.SetYesNo(0);
Input_Enabled.SetDescription("This input enables the study and allows it to function. Otherwise, it does nothing.");

// This is false by default. Orders will be simulated.
sc.SendOrdersToTradeService = false;

sc.AllowMultipleEntriesInSameDirection = false;

//This must be equal to or greater than the order quantities you will be submitting orders for.
sc.MaximumPositionAllowed = 10000;

sc.SupportReversals = true;
sc.AllowOppositeEntryWithOpposingPositionOrOrders = true;

// This variable controls whether to use or not use attached orders that are configured on the Trade Window for the
chart.
sc.SupportAttachedOrdersForTrading = false;

//This variable controls whether to use the "Use Attached Orders" setting on the Trade Window for the chart
sc.UseGUIAttachedOrderSetting = false;

sc.CancelAllOrdersOnEntriesAndReversals = false;
sc.AllowEntryWithWorkingOrders = false;
sc.CancelAllWorkingOrdersOnExit = true;
sc.AllowOnlyOneTradePerBar = false;

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

sc.AutoLoop = true;
sc.GraphRegion = 0;

return;
}

```

//These must be outside of the sc.SetDefaults code block since they have a dependency upon an actual chart object

existing

```
sc.SupportTradingScaleIn = 0;
sc.SupportTradingScaleOut = 0;

if (!Input_Enabled.GetYesNo())
    return;

if (sc.LastCallToFunction)
    return; //nothing to do

int& PriorOrderFillEntrySize = sc.GetPersistentInt(1);

int CurrentOrderFillEntrySize = sc.GetOrderFillArraySize();

if (CurrentOrderFillEntrySize != PriorOrderFillEntrySize)
{
    PriorOrderFillEntrySize = CurrentOrderFillEntrySize;
    if (CurrentOrderFillEntrySize > 0)
    {
        s_SCOrderFillData OrderFillData;

        sc.GetOrderFillEntry(CurrentOrderFillEntrySize - 1, OrderFillData);
        SCString OrderFillMessage;

        uint32_t OrderQuantity = static_cast<uint32_t>(OrderFillData.Quantity);

        OrderFillMessage.Format("New order fill: InternalOrderID = %u, FillPrice = %f, Quantity = %u",
            OrderFillData.InternalOrderID, OrderFillData.FillPrice, OrderQuantity);

        sc.AddMessageToLog(OrderFillMessage, 0);

        s_SCPositionData SCPositionData;

        if (sc.GetTradePosition(SCPositionData))
        {
            SCString PositionDataMessage;

            PositionDataMessage.Format("Position Data: AveragePrice = %f, AllWorkingBuyOrdersQuantity = %d,
            AllWorkingSellOrdersQuantity = %d", SCPositionData.AveragePrice, static_cast<int>
            (SCPositionData.AllWorkingBuyOrdersQuantity), static_cast<int> (SCPositionData.AllWorkingSellOrdersQuantity));

            sc.AddMessageToLog(PositionDataMessage, 0);
        }
    }
}

/*=====*/
SCSFExport scsf_TradingExampleUnmanagedAutomatedTrading(SCStudyInterfaceRef sc)
{
    // Define references to the Subgraphs and Inputs for easy reference
    SCSubgraphRef Subgraph_BuyEntry = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellEntry = sc.Subgraph[1];
    SCInputRef Input_Enabled = sc.Input[0];

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

        sc.GraphName = "Trading Example: UnmanagedAutomatedTrading";
    }
}
```

```

Subgraph_BuyEntry.Name = "Buy Entry";
Subgraph_BuyEntry.DrawStyle = DRAWSTYLE_ARROW_UP;
Subgraph_BuyEntry.PrimaryColor = RGB(0, 255, 0);
Subgraph_BuyEntry.LineWidth = 2;
Subgraph_BuyEntry.DrawZeros = false;

Subgraph_SellEntry.Name = "Sell Entry";
Subgraph_SellEntry.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_SellEntry.PrimaryColor = RGB(255, 0, 0);
Subgraph_SellEntry.LineWidth = 2;
Subgraph_SellEntry.DrawZeros = false;

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

sc.AllowMultipleEntriesInSameDirection = true;
sc.MaximumPositionAllowed = 3;
sc.SupportReversals = false;

// This is false by default. Orders will go to the simulation system always.
sc.SendOrdersToTradeService = false;

sc.AllowOppositeEntryWithOpposingPositionOrOrders = true;

sc.SupportAttachedOrdersForTrading = false;

sc.CancelAllOrdersOnEntriesAndReversals = false;
sc.AllowEntryWithWorkingOrders = true;
sc.CancelAllWorkingOrdersOnExit = false;

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

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

sc.AutoLoop = 0;
sc.GraphRegion = 0;

return;
}

if (!Input_Enabled.GetYesNo())
return;

int& r_MenuID = sc.GetPersistentIntFast(1);

if (sc.LastCallToFunction)
{
    sc.RemoveACSCChartShortcutMenuItem(sc.ChartNumber, r_MenuID);

    return;
}

if (sc.IsFullRecalculation != 0)
{
    // Add chart shortcut menu item if not already added
    if (r_MenuID <= 0)
    {
        r_MenuID = sc.AddACSCChartShortcutMenuItem(sc.ChartNumber, "Execute auto trade");
    }
}

```



```

}

// wait for an event
if (sc.MenuEventID != 0)
{
    if (sc.MenuEventID == r_MenuID)
    {
        s_SCNewOrder NewOrder;
        NewOrder.OrderQuantity = 3;
        NewOrder.OrderType = SCT_ORDERTYPE_MARKET;
        //NewOrder.Price1 = 0;
        NewOrder.Target1Offset = 50 * sc.TickSize;
        NewOrder.TimeInForce = SCT_TIF_GOOD_TILL_CANCELED;
        NewOrder.Stop1Offset = 50 * sc.TickSize;

        sc.BuyEntry(NewOrder);

        s_SCNewOrder ExitOrder;
        ExitOrder.OrderQuantity = 1;
        ExitOrder.OrderType = SCT_ORDERTYPE_MARKET;

        sc.SellEntry(ExitOrder);
    }
}
}

```