User Guide

Sample Order Tickets

Snaggs

Contents

1 Introduction

1.1 Purpose

OrderTickets offer a more controlled way to place orders in TradeStation, but can often be difficult and lengthy to code when creating complex orders. This document is intended to guide you through an easy way to create simple to complex orders. Order types range from a simple Market order to Breakout orders that send stops and targets based on the breakout direction.

The examples in the application can be used to create orders for Equities, Stock Options, Futures and Index Options. Logging is provided by the ELCFormat functions created by TradeStation.

1.2 Goals of this effort

- Create an easy way for developers just getting into OOEL to create orders
- Reduce the lines of code needed to create an order
- Simplify the order creation process
- Create a function that can be used in any study
- Provide examples how orders from Trade Bar can be created using OrderTickets
- Create a way to quickly look at an OrderTicket to see what it's doing
- Standardize OrderTicket creation to easily add to OSO/OCO and Bracket orders
- Provide logging to be able to see the OrderTypes that are created and see how they interact with the different data providers and event handlers

1.3 Types of orders that can be created

- Single ticket orders
- Entries and Exits using Market, Limit, Stop and Trailing Stop orders
- OSO/OCO, OSO/OCO/Bracket
- Breakouts, Breakout/Fade
- Orders with Time Activation and Deactivation rules
- Order with Price Activation rules
- Examples for Long and Short entries and exits in different configurations

1.4 What's not covered

- Some auto price settings like HitOrTake, IfTouchPlus, Improve, Join, Shave, Split
- Moving stops to breakeven
- Replacing OrderTickets
- Reversing positions
- Handling partially filled orders

2 Installation

This section details the files inside the ELD and the steps to install the indicator from the ELD.

2.1 Files included in the ELD

ELCFormatGeneratingApplications	Function
ELCFormatOrder	Function
ELCFormatOrderTicket	Function
ELCFormatPosition	Function
IffString	Function
LogEvent	Function
OTAddActivationRulePrice	Function
OTAddActivationRuleTime	Function
OTCreate	Function
Sample Order Tickets	Indicator

2.2 Naming Conventions

Naming conventions help determine the scope of the variables and where they're defined. Having a good naming convention helps avoid name collisions where a variable matches a reserved word or other variable at a higher level scope. I've found the following naming convention works well.

- Inputs start with a lowercase i, for input: iShowMsgs.
- Constants are all caps: APPVERSION.
- Global variables are Pascal case: LongBrackets.
- User created method parameters start with a lowercase p, for parameter: pAccount.
- System generated methods for event handlers have the default style they're created with.
- Local variables in methods are camel case: otEntry.
- Dictionary/AppStorage keys are camel case: ["printMsgs"].
- Form controls are camel case and have a 3 char prefix for their control type: cboOrderType.

2.3 Steps to install the indicator

- Extract the Sample Order Tickets.ELD file from the Zip file.
- Double click it to install it.
- Take note when installing the indicator. Some files may already exist on your system and it's up to you to determine if you should overwrite them.

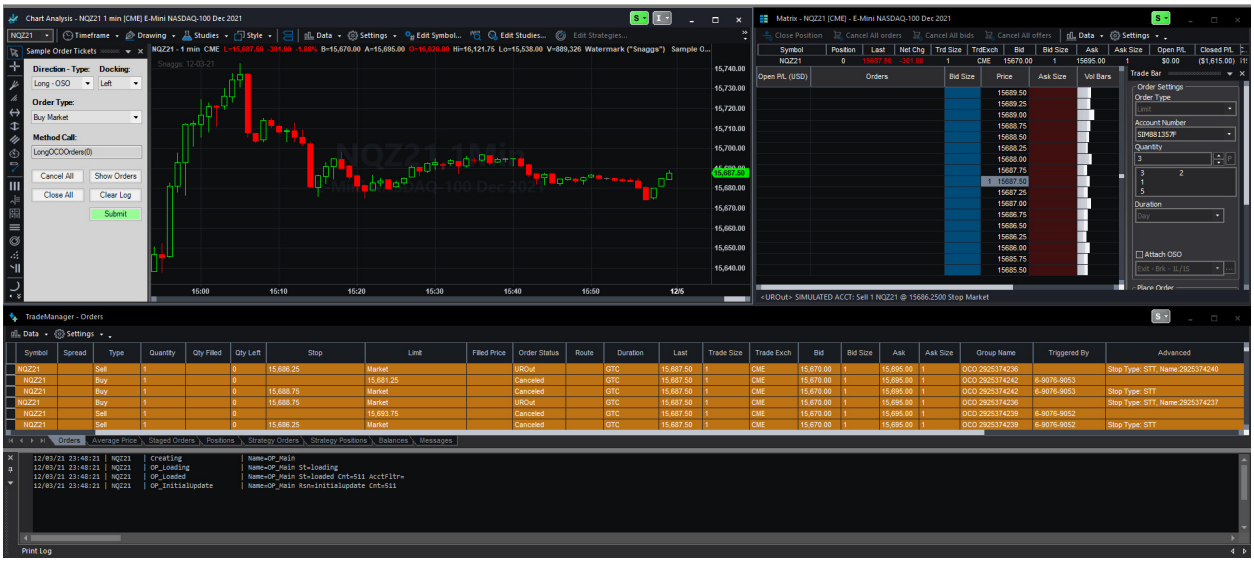
3 Configuration

This section goes over getting the most out of the indicator and how to configure it

!!! BE SURE TO USE A SIM ACCOUNT WHEN YOU USE THIS INDICATOR !!!

3.1 The Workspace

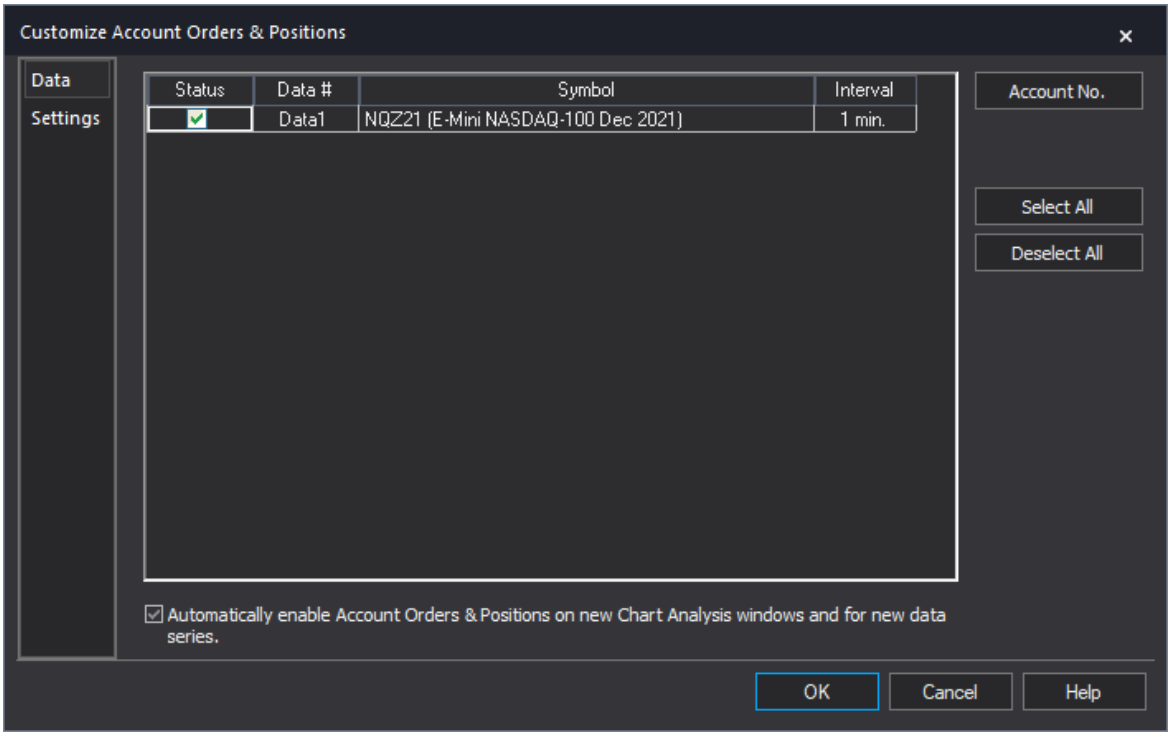
To get the most out of the indicator, it's good to see the Chart, Matrix, TradeManager and the PrintLog window. The image is a bit small, but gives you an idea of the layout. A workspace of this is included in the package and might be a quicker way to get up and running.



3.2 Customize the Chart to show the orders

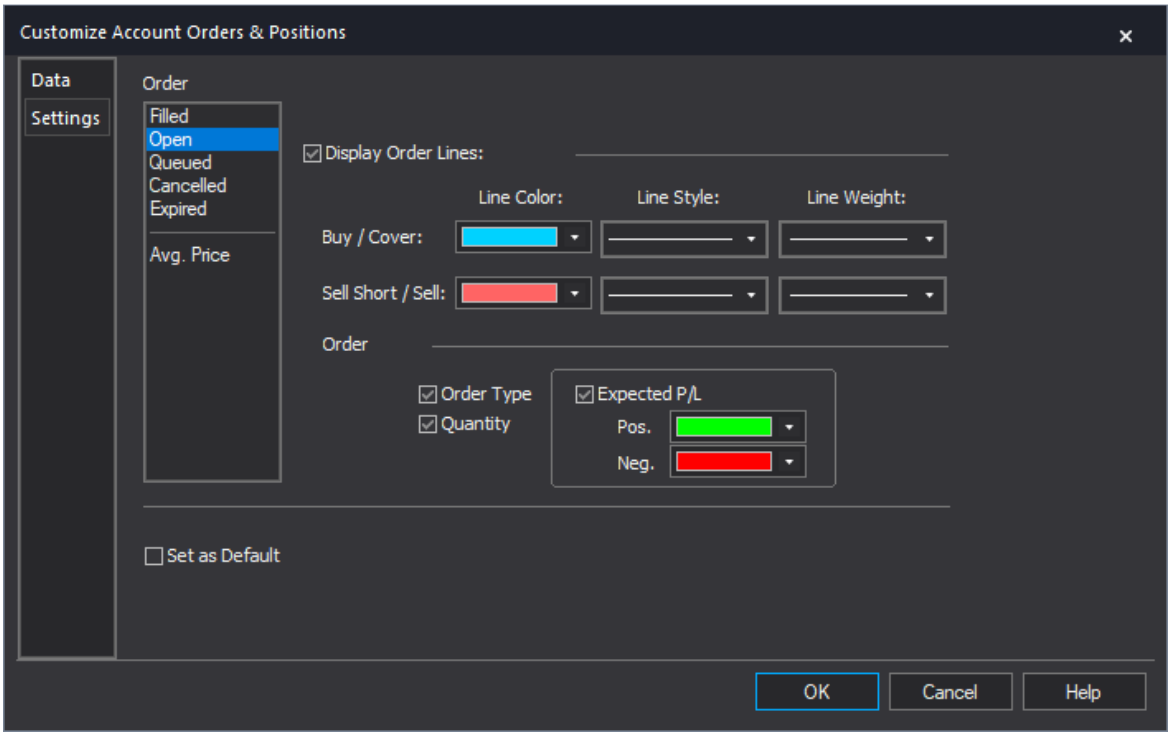
The chart should have the following settings already configured, but if not, here's how they should be set. To get to the settings, right click the chart and select Settings -> Account Orders & Position.

On the Data tab, make sure Status is checked for Data1.

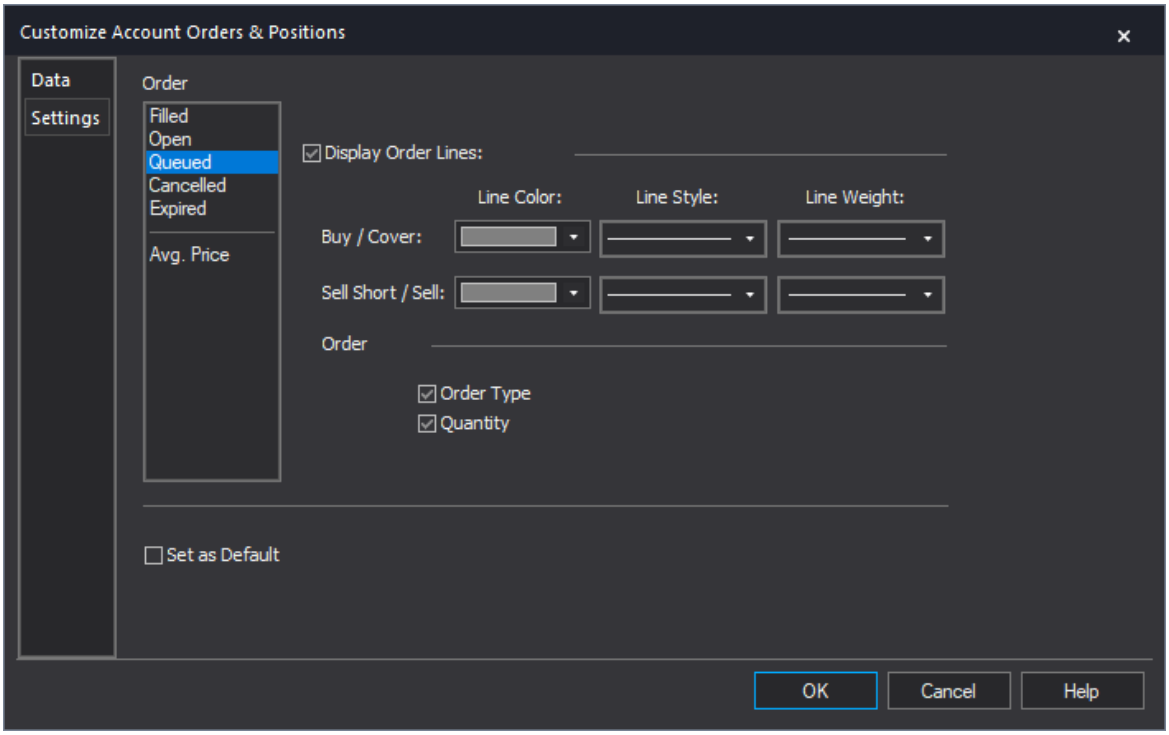


Click the Settings tab. Filled should be unchecked.

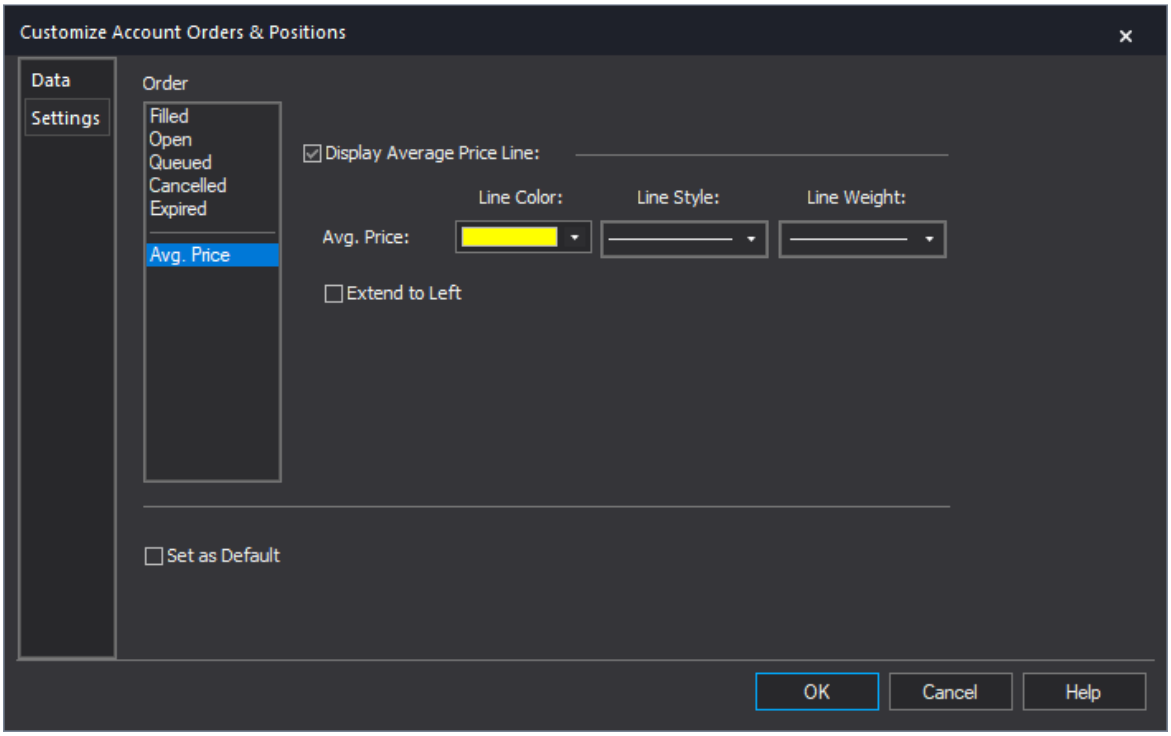
Open should have Display Order Lines checked.



Queued should have Display Order Lines checked.



Avg. Price should have Display Average Price Line checked.



This will ensure that orders will show up on the chart to help you visualize what the order looks like.

3.3 Indicator Customization

Open the customization for the indicator and be sure “Enable order placement objects” is checked.

There are 3 inputs for logging. When logging is enabled, the indicator will print information to the PrintLog window based on areas of the app that do logging. Areas are considered OrderTicket, OrdersProvider, PositionsProvider, etc.

ShowMsgs: This globally enables and disables ALL logging.

ShowOrderTickets: This enables and disables logging only for OrderTickets.

ShowOrdersProvider: This enables and disables logging only for the OrdersProvider.

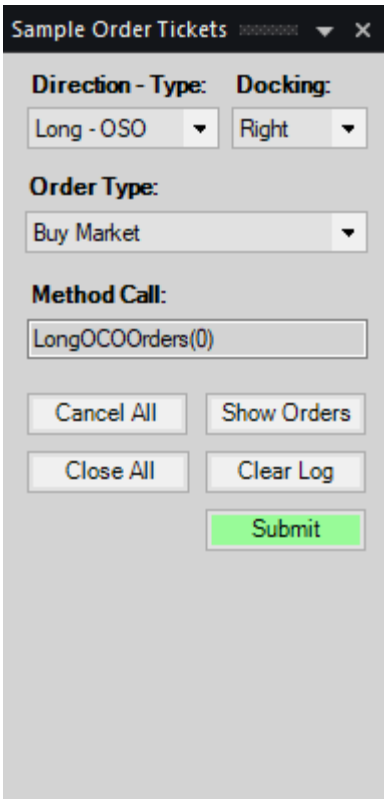
ShowPositionsProvider: This enables and disables logging only for the PositionsProvider.

Logging is enabled by default and you’ll see messages printed to the PrintLog window as they occur in real-time.

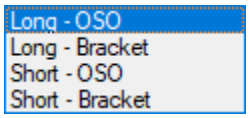
4 The Interface

This section will go over the different parts of the interface.

4.1 Form Layout



Direction-Type: Each item represents a family of related order types for a specific direction.



Docking: Allows you to dock the form to the left or right of the chart.

Order Type: This contains a list of trade types that can be created.

Method Call: This tells you where to find the code in the indicator to see how the order was created. The name of the function and the parameter used to call it are shown here.

Cancel All: Cancels all open orders.

Close All: Closes all open positions.

Show Orders: Shows all the orders in the orders provider.

Clear Log: Clears the PrintLog window.

Submit: Submits the selected order.

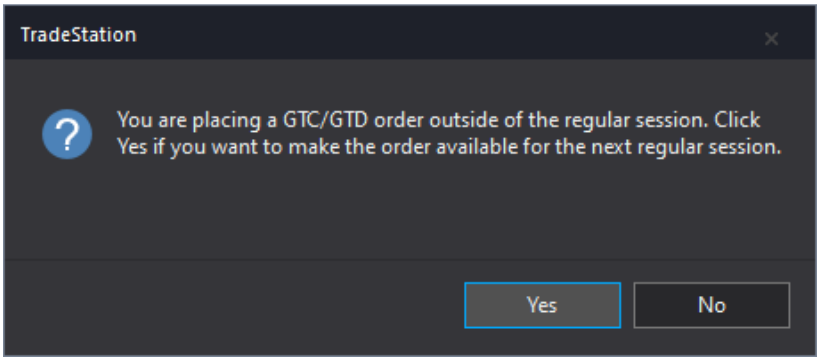
5 Using the Indicator

I've found it best to test these orders on Futures like ES. You must use a tradable symbol like ESZ21 so the indicator can make trades. A continuous contract like @ES won't work. If you're not subscribed to Futures data, you can still test the indicator on delayed data or after hours.

I found after hours and slow periods are nice to test with because they allow you to look at the TradeManager to see how the order was constructed before it gets filled. I've tried to use predefined settings that work well with all symbol types (Stocks, Options and Futures). The preset values for the more advanced trades are set to give you time to look at the order before it fills. For some orders, the content in the Stop and Limit box will change after the order fills.

5.1 The First Trade

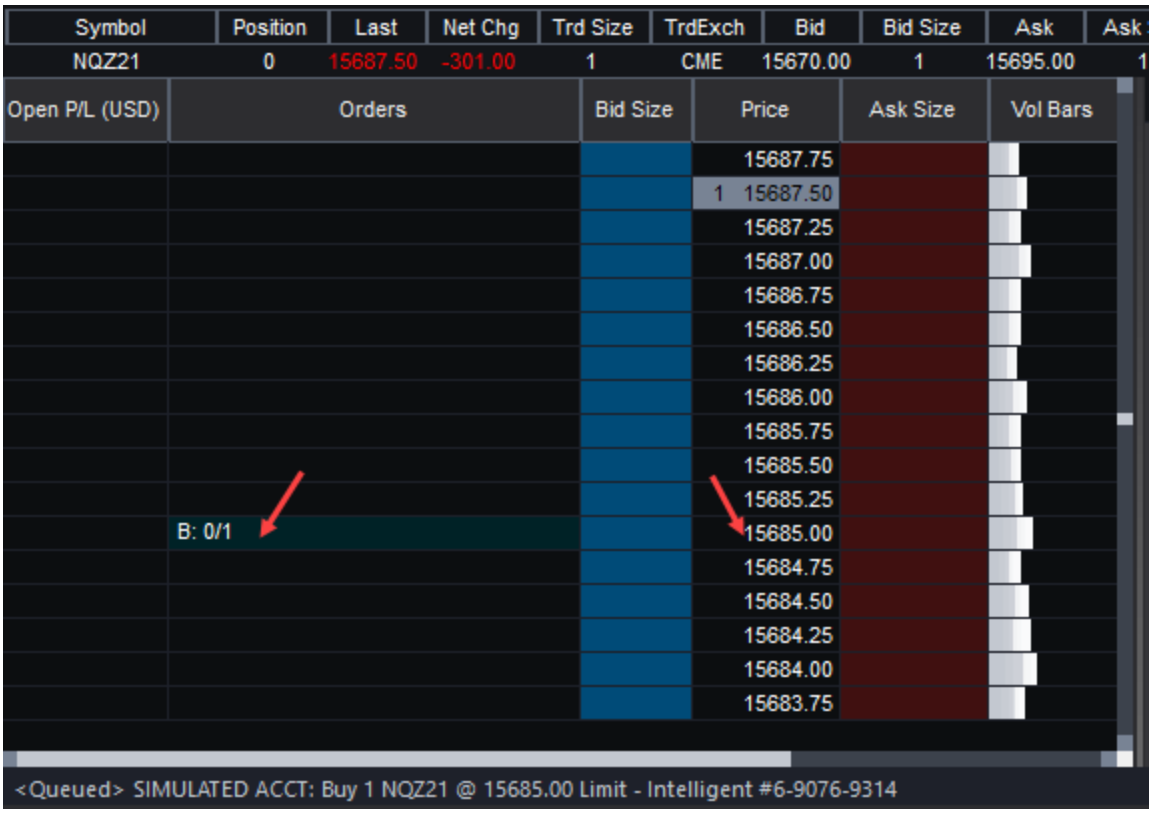
Select 'Long - OSO' from 'Direction - Type' and 'Buy Limit Offset' from the 'Order Type'. Then click Submit. If you're trading after hours you may get this popup message. Click Yes to place the trade.



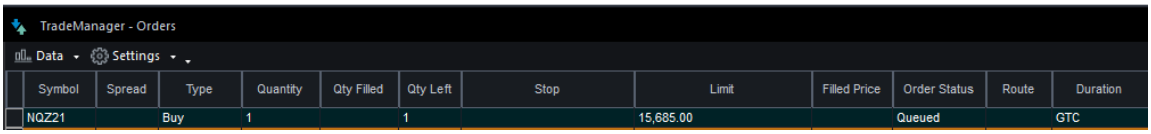
5.2 Reviewing the results of the first trade

The chart will display a gray line showing that the order is Queued if the market is closed. It'll show Received if the market is open. A line will display at the limit price indicated by the gray price to the right. The L indicates it's a LIMIT order. The 1 indicates quantity of contracts/shares.

The Matrix will show 1 buy order at the LIMIT price. The B is for Buy. 0/1 means 0 contracts/shares filled of 1 total.

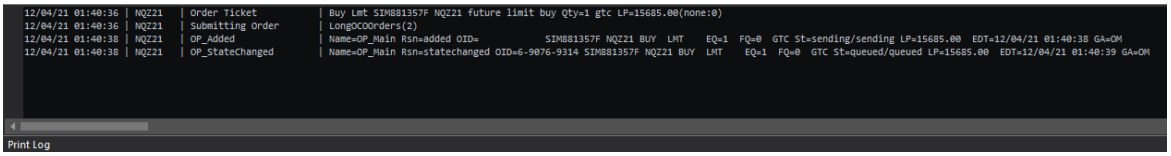


The TradeManager is your best friend and will show you the structure of the ticket(s) that were sent to the Trade Server. Here it shows the LIMIT order at 15685.00.



The PrintLog displays messages from the different objects that send messages to it.

- Column 1: DateTime
- Column 2: Chart Symbol
- Column 3: 'Action' that cause the log event
- Column 4: Details about the 'Action'



5.3 Breaking down the Log Data

Row 1:

Order Ticket | Buy Lmt SIM881357F NQZ21 future limit buy Qty=1 gtc LP=15685.00(none:0)

The Action is 'Order Ticket' meaning that code related to an OrderTicket created this row, followed by a separator bar |.

The name of the order is 'Buy Lmt' on a specific Sim account for symbol NQZ21 which is a Category or SecurityType of 'future'. The order specifics are a limit buy order for a quantity of 1. Duration is gtc (good till cancelled) and the Limit Price (LP) = 15685.00. (none:0) means LimitPriceStyle for the OrderTicket was set to none. The 0 is the value of the LimitPriceOffset value on the OrderTicket.

All this information comes from the ELCFormatOrderTicket function provided by TradeStation. You can open the function if you want to see what each value means. They can be somewhat cryptic at first, but over time they get easier to read.

Row 2:

Submitting Order | LongOCOOrders(2)

This comes from the indicator and tells you what function was called 'LongOCOOrders' along with the parameter used to create the order, '2'. There are several order types in a switch statement in the code to place different orders. This is telling you where to look in the code for the details about the order that was placed. Here is the method that was called.

```
249
250 //------------------------------------------------------------------------------
251 //   Long OCO orders method
252 //------------------------------------------------------------------------------
253 #region LongOCOOrders
254 method Orders LongOCOOrders(string pAccount, string pSymbol, SecurityType pSecurityType, int pType)
255 var:
256     DateTime dt,
```

The 4 methods that create orders all receive the same 4 parameters as seen above. The p prefix is part of my naming convention to identify parameters that are passed to a method.

Here is the Case value (2) that was used to create the order along with the code for the order.

```
case 2: // Verified
    // Entry - Single Order - Long
    // Buy Limit offset 10 ticks
    value1 = OTCreate("Buy Lmt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "lmt", "gtc", -10, otEntry);
    ords = otEntry.Send(true);
```

The code calls the function OTCreate() and passes several parameters to create a single OrderTicket. The order of the parameters was chosen so reading it from left to right is a natural progression of building the order. The most important parameters were placed first and are more of a macro level. As we go to the right the parameters become more micro level. The last parameter is the OrderTicket that is returned from the function.

Functions can't return object types, they can only return primitive types like, bool, double, int, string, etc. The function must return a value. If the function succeeded, a 0 is returned and captured in value1. For this indicator, the value returned is not used, but it still needs to be captured.

6 The OTCreate Function

This is the central focus of this indicator and this function can be called by any study or strategy to create an OrderTicket.

6.1 OTCreate Inputs

All the inputs for a function are prefixed with a lowercase i for input.

- iName: The name of the OrderTicket
- iAccount: The account number to place the order against
- iSymbol: the Symbol to be traded
- iSecurityType: The security type that's being traded
- iAction: Buy, Sell, etc
- iQuantity: The quantity of shares or contracts to trade
- iPrimaryValue: This is the base value price for the OrderTicket
- iOrderType: Limit, Market, etc
- iDuration: "Day", "GTC", etc
- iSecondaryValue: This is an offset value from the PrimaryValue for the OrderTicket
- oOrdTicket: This is the OrderTicket object that is returned to the calling function. This object will have all the properties set on it based on the input values passed to the function.

```
33
34  input:
35      string iName(StringSimple),
36      string iAccount(StringSimple),
37      string iSymbol(StringSimple),
38      SecurityType iSecurityType(ObjectSimple),
39      string iAction(StringSimple),       // Buy, BuyToCover, Sell, SellShort
40      int iQuantity(NumericSimple),
41      double iPrimaryValue(NumericSimple),
42      string iOrderType(StringSimple),    // Limit, Market, StopLimit, StopMarket
43      string iDuration(StringSimple),     // DAY, FOK, GTC
44      double iSecondaryValue(NumericSimple),
45      OrderTicket oOrdTicket(ObjectRef);
46
```

I suggest looking at this function to understand how it works. Each OrderType has at least two labels you can use to create the ticket. One is the short form, like "MKT" the other is the long form "MARKET". Based on how you want to build the ticket, the correct properties will be set.

Note: When first creating tickets, it's still possible to setup the ticket in such a fashion that the Trade Server won't accept it. An example is creating GTC+ orders in after hours. The creation of the ticket will be fine, but the Trade Server won't accept it.

6.2 Ticket Creation, Logging and AppStorage

```
switch pOrderType.Trim().ToUpper()
begin
    // Market order
    case "MKT", "MARKET":  // Verified
        ordTicket.Type = OrderType.Market;
        ordTicket.LimitPrice = 0;
        ordTicket.LimitPriceStyle = PriceStyle.None;
        ordTicket.LimitPriceOffset = 0;
        ordTicket.StopPrice = 0;
        ordTicket.StopPriceStyle = PriceStyle.None;
        ordTicket.StopPriceOffset = 0;
        break;

    // Limit order
    case "LMT", "LIMIT":  // Verified
        ordTicket.Type = OrderType.Limit;
        ordTicket.LimitPrice = pPrimaryValue + pSecondaryValue * tickSize;
        ordTicket.LimitPriceStyle = PriceStyle.None;
        ordTicket.LimitPriceOffset = 0;
        ordTicket.StopPrice = 0;
        ordTicket.StopPriceStyle = PriceStyle.None;
        ordTicket.StopPriceOffset = 0;
        break;
```

There is some math in OTCreate function that converts ticks to points based on which parameter the value comes in on. For the most part, as a developer, you will rarely need to look at this code.

Since the OrderTicket is created in the OTCreate function using the 'new' keyword, there is no reason to create the ticket outside of the function. You just have to have a variable to receive the newly created OrderTicket.

There is logging at the end of this function. It uses AppStorage to be able to determine what it should print to the PrintLog. If values related to it are not found in AppStorage, it will print a warning in the PrintLog. This application is setup to have the correct values for logging, so you shouldn't see these warning messages. This is mostly used when creating a new indicator to help get the logging variables defined.

For most tickets, the PrimaryValue is in POINTS, and the SecondaryValue at the end is in TICKS.

```
case 2: // Verified
    // Entry - Single Order - Long
    // Buy Limit offset 10 ticks
    value1 = OTCreate("Buy Lmt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "lmt", "gtc", -10, otEntry);
    ords = otEntry.Send(true);

case 3: // Verified
    // Entry - Single Order - Long
    // Buy Stop Market offset 5 ticks
    value1 = OTCreate("Buy StpMkt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "stpmkt", "gtc", 5, otEntry);
    ords = otEntry.Send(true);
```

Most tickets are setup this way, but some are both in points, some are both in ticks.

PrimaryValue

SecondaryValue

7 OrderTicket Properties Explained

The OrderTicket class has quite a few properties on it. The goal of the OTCreate function is to simplify the OrderTicket creation process. Before we get into some of the more complex orders, it's important to understand some of the terminology used to build orders.

The naming used in the indicator does not use Target, Stop or Stop Loss. It uses Limit, Stop and StopLimit. Based on this, you can determine which one is the Target order and which one is the Stop order. Sometimes these are used for entry orders too. In the following Terms section there are order types that have the Plus term in them. These are order types that usually use the SecondaryValue in Ticks as an offset from the PrimaryValue base price in points.

A simple way to keep them straight is if the value is absolute, like $50, then it's in points. If it's an offset, then it's in Ticks. For OrderTypes that use 2 absolute prices, the Primary and Secondary Values are in Points. If the OrderType uses 2 offset prices, the Primary and Secondary Values are in Ticks. You'll get the hang of this when you see enough orders in the examples.

7.1 Terms

- **MKT**: This OrderType is a market order that will be placed at the current bid or ask based on direction.
  PrimaryValue = 0
  SecondaryValue = 0

- **LMT**: This OrderType is a limit order. Limit orders wait for the price to come to them to be filled.
  PrimaryValue = Limit Price in Points
  SecondaryValue = Offset from current price in Ticks

- **LMT-PP**: This is a ParentPlus OrderType. This links the Limit order to the parent order fill price. Depending on the entry type, slippage can occur. When a secondary order is linked to the parent order it preserves the offset. For example a long market order is placed while the price is at 50. The Limit order is for 20 ticks above the intended fill at 50. Due to slippage, the price may get filled at 50.03. This gives a 3 tick loss of intended profit. If a ParentPlus OrderType is used, the Limit order will be set to exit 20 ticks above 50.03 at 50.23. ParentPlus type orders preserve the offset from the parent order fill price to the Limit order price.
  PrimaryValue = Limit price in Points
  SecondaryValue = Offset from Parent fill in Ticks

- **STPLMT**: This OrderType can activate a limit order to buy or sell when a specific stop price has been met. The STOP price is always closest to the current price. The LIMIT price is always at or further away. Both LIMIT and STOP prices are absolute prices, so the PrimaryValue and SecondaryValue are both in Points.
  PrimaryValue = Stop price in Points
  SecondaryValue = Limit price in Points

- **STPLMT-SP**: This is a StopPlus OrderType. When StopPlus is used, the STOP price is the PrimaryValue in Points, and the LIMIT order is offset in Ticks from the STOP Price.
  PrimaryValue = Stop price in Points
  SecondaryValue = Limit price offset from Stop price in Ticks

- **STPLMT-PP-SP**: This OrderType preserves the offset between the STOP offset and the Parent order fill price (PP). The SP preserves the price offset between the STOP price and the LIMIT price offset. Both STOP and LIMIT prices are offset prices, so the PrimaryValue and SecondaryValue are both in Ticks.
  PrimaryValue = Stop offset from Parent fill in Ticks
  SecondaryValue = Limit offset from Stop price in Ticks

- **STPMKT**: This OrderType is a market order that gets filled when the price is hit. An offset value in Ticks can be used to change where the STPMKT order is placed.
  PrimaryValue = Stop price in Points
  SecondaryValue = Offset from current price in Ticks

- **STPMKT-PP**: This OrderType ignores the PrimaryValue but uses the SecondaryValue as price offset from the Parent fill price in Ticks.
  PrimaryValue = 0
  SecondaryValue = Stop offset from Parent fill in Ticks

- **TSTP**: This OrderType is a Trailing Stop. The PrimaryValue is ignored and the trailing amount is in ticks. If you want the trailing stop in %, then you can override the TrailingStopBehavior and adjust the TrailingStopAmount once the OrderTicket is returned. Doing this is not covered in the application or this documentation.
  PrimaryValue = 0
  SecondaryValue = Offset from current price in Ticks

8 Creating OrderTickets with OTCreate

In this section we'll look at how to create an OrderTicket using the OTCreate function. All of the orders created in this app can be created from the Trade Bar in TradeStation. Long - OSO has the most tickets in since it shows a lot of different order types. I've duplicated some of them in the Short - OSO section to show what the Short version looks like.

There should be enough order type examples that you can piece together the order you want if it's not included here. This indicator is a building block to show you different techniques used to build different types of orders. It's not intended to be all encompassing.

Here's a few order examples. The first comment below the case statement explains what the order is. The line below that gives the details of the order.

8.1 Simple Buy Orders

Case 4: Is a buy stop limit order. We have to set the stop price and the limit price. The Stop Price is first for the PrimaryValue. It's Close + 10 * TickSize. This gets a result Points. The Limit price is the SecondaryValue. It's also Close + 15 * TickSize to get the result in Points. We could use the format Close + 2.5 for the Stop and Close + 3.75 for the Limit. Either way works for Point values.

Case 5: Is a buy stop limit Stop Plus order. Here I used Close + 2.5 points, but the offset for the Limit order, from the Stop order is 5 ticks.

Case 6: Is a trailing buy stop order. The PrimaryValue is set to 0 since the Trailing Stop starts to trail from the current price. The PrimaryValue is ignored in the ticket creation process.

```
case 4: // Verified
    // Entry - Single Order - Long
    // Buy Stop Close + 10 ticks, Limit Close + 15 ticks
    value1 = OTCreate("Buy StpLmt", pAccount, pSymbol, pSecurityType, "buy", 1, Close + 10 * tickSize, "stplmt", "gtc", Close + 15 * tickSize, otEntry);
    ords = otEntry.Send(true);

case 5: // Verified
    // Entry - Single Order - Long
    // Buy Stop Close + 2.5 points, Limit StopPlus offset 5 ticks
    value1 = OTCreate("Buy StpLmt SP", pAccount, pSymbol, pSecurityType, "buy", 1, Close + 2.5, "stplmt-sp", "gtc", 5, otEntry);
    ords = otEntry.Send(true);

case 6: // Verified
    // Entry - Single Order - Long
    // Buy Trailing Stop 12 ticks (Trailing stops are in ticks and always positive)
    value1 = OTCreate("Buy TStp", pAccount, pSymbol, pSecurityType, "buy", 1, 0, "tstp", "gtc", 12, otEntry);
    ords = otEntry.Send(true);
```

8.2 Simple OSO Orders

Case 10: Is a buy Limit order with a sell stop offset of 15 ticks. You have to get the sign correct on the offset or the stop may fill as soon as the buy order fills. We're going to create 2 OrderTickets for this order. They are offset to show how I'm going to link them. The top order is the primary ticket and the indented one under it, is the secondary ticket that'll be used in the OSO ticket.

First we create the entry order on line 331 as a buy limit order at the close and 0 offset. The OrderTicket comes back to us through the otEntry variable at the end.

Next we create the stop order on line 332 as a sell stop market with a -15 tick offset below the current price. The OrderTicket comes back to us through the otStop variable at the end. Note that the 15 tick gap may not be persevered if the limit order gets a better fill than the current price. This is because we're not using a ParentPlus order type.

Next we create an OSOOrderTicket instance. We assign the otEntry to the PrimaryTicket and add the otStop as a SecondaryTicket. Then we send the order. Since this is an OSO (One Sends Other) order type, when the entry fills, the stop will be sent.

```
328         case 10: // Verified
329             // Entry - Single Order - Long with Exit - Stop Only
330             // Buy Limit, Sell Stop Market offset 15 ticks
331             valuel = OTCreate("Buy Lmt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "lmt", "gtc", 0, otEntry);
332                 valuel = OTCreate("Sell StpMkt", pAccount, pSymbol, pSecurityType, "sell", 1, Close, "stpmkt", "gtc", -15, otStop);
333
334             // OSO group for entry and exit tickets
335             oso = new OSOOrderTicket();
336             oso.PrimaryTicket = otEntry;
337             oso.SecondaryTickets.Add(otStop);
338
339             ords = oso.Send(true);
340
341         case 11: // Verified
342             // Entry - Single Order - Long with Exit - Stop Only
343             // Buy Limit offset 5 ticks, Sell Stop Market ParentPlus offset 15 ticks from parent fill
344             valuel = OTCreate("Buy Lmt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "lmt", "gtc", -5, otEntry);
345                 valuel = OTCreate("Sell StpMkt PP", pAccount, pSymbol, pSecurityType, "sell", 1, 0, "stpmkt-pp", "gtc", -15, otStop);
346
347             // OSO group for entry and exit tickets
348             oso = new OSOOrderTicket();
349             oso.PrimaryTicket = otEntry;
350             oso.SecondaryTickets.Add(otStop);
351
352             ords = oso.Send(true);
353
```

Case 11: Is a buy limit order 5 ticks below the current price. This is almost the same order as above, but this one is using a tick offset from the current price (Close).

The Stop ticket is created below that as a Stop Market ParentPlus ticket. This is almost the same ticket as Case 10, except we're setting the stop 15 points below the parent fill price. This order will preserve the 15 tick gap, where the one above doesn't.

We build the order the same way as above and send it.

For Case 10, notice the Stop price and Limit price are absolute prices.

Data Settings

Symbol	Spread	Type	Quantity	Qty Filled	Qty Left	Stop	Limit	Filled Price	Order Status
NQZ21		Buy	1		1		15,687.50		Queued
NQZ21		Sell	1		1	15,683.75	Market		Queued

For Case 11, the Stop price uses ParentPlus and shows Offset from Primary Fill -3.75. There are a couple things to observe here. First, both the Stop and Limit price are displayed in points, even though we set them in ticks. Also, the Limit price is 5 ticks (1.25 pts) below where the limit price was for the above order. Everything appears to working.

Data Settings

Symbol	Spread	Type	Quantity	Qty Filled	Qty Left	Stop	Limit	Filled Price	Order Status
NQZ21		Buy	1		1		15,686.25		Queued
NQZ21		Sell	1		1	Offset from Primary Fill -3.75	Market		Queued

8.3 Handling OrderTicket Events

At the end of each method that creates an order, is a small section of code that adds each OrderTicket to an event handler. This lets us track the progress of the order in the PrintLog window.

```
// Assign each OrderTicket to the Order_Updated event handler
for idx = 0 to ords.Count - 1
begin
    ords[idx].Updated += Order_Updated;
end;

return ords;
```

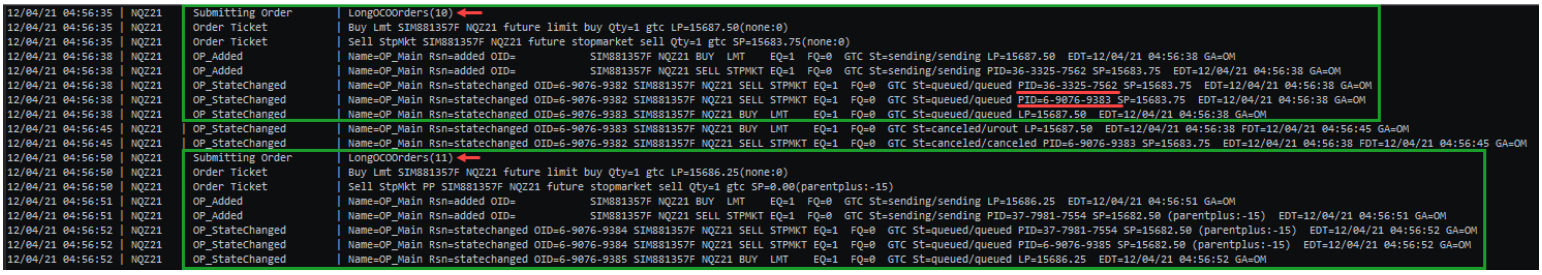
A bare-bones Order event handler was used for this application to capture all updates to a ticket.

```
//--------------------------------------------------------------------------------
//   Order updated event handler
//--------------------------------------------------------------------------------
#region OrderEventHandler
method void Order_Updated(object sender, OrderUpdatedEventArgs args)
var:
    Order ord,
    string msg;
begin
    ord = sender astype Order;

    msg = string.Format("Rsn={0} {1}",
     args.Reason.ToString(),
     ELCFormatOrder(ord, 2, false));

    value1 = LogEvent(PrintMsgs and PrintOT, "Order_Updated", msg);
end;
#endregion
```

8.4 Simple OSO Order PrintLog Analysis

The first order, order 10, shows up in the top green box. The red arrow is pointing to the method call that was made with the parameter of 10. We see 2 OrderTicket actions that represent the Buy and Sell orders. Since both of these orders were absolute prices, there was no price offset for them as indicated at the end of the 2 OrderTicket Lines by (none: 0).

Sometimes the OrdersProvider will change the PID on the ticket as indicated by the 2 red lines on the first order. It did it on the second order too, but I didn't highlight it.



The text between the two boxes is the cancellation of the first order before I placed the second order. The second order, order 11, shows up in the bottom green box. The red arrow is pointing to the method call that was made with the parameter of 11. Again 2 OrderTicket actions create the Buy and Sell orders. For this Sell order we used a ParentPlus order type. At the end of the order we notice (parentplus: -15). This value is represented in ticks.

9 Activation Rules - Time and Price

In this section we go into activation rules for time and price. There are activation and deactivation rules for time based rules but only activation rules for price. They're all covered in this indicator. Activation rules are added as an add-on to the OrderTicket. The OrderTicket is created, then passed to another function to have the rule added to it. I call this a form of decorating the OrderTicket with an Activation Rule.

A few things to know about Activation Rules.

- Time based rules are for 'Day' duration only. If you want the state to go to 'Expired', then 'Day' duration should be used. If you use something else, they will change to 'Queued' when they expire. This can be confusing when you see it in the TradeManager.
- Activation rules only work when the session is open.
- Time based rules start when the order is placed, NOT from when a parent order fills.
- Currently time deactivation rules only work on Futures, not stocks, stock options or index options, although they should work on those too. If you test this, it has to be on a futures contract. At least until it's fixed.

For the time activation rules, the options are activate, deactivate, absolute and relative.

Absolute time is in ELTime format (HHMM no seconds) like 1344 for 1:44pm. With absolute time you can create orders that buy or sell at the open or close positions before the market closes.

Relative time is in seconds. To convert to minutes or hours, multiply it up. To get to 6 mins this can be use: 6 * 60. For 2 hours, use 2 * 60 * 60. This is clearer when you look at it than seeing 7200.

Let's see how to use them.

9.1 Activate - Absolute

This order creates a buy limit order and a sell market order. The sell order has an activation rule applied to it that uses an absolute value in ELTime format that tells the sell order when to execute. This activation rule tells the sell order to execute at 1459. The lowest time interval is 1 min for absolute time rules.

```
533         case 40: // Verified
534             // Entry - Single Order - Long with Exit
535             // Absolute time activation rule
536             // Time is in ELTime (HHMM, no seconds)
537             // Activate the Sell Stop at the end of the day 1459
538             value1 = OTCreate("Buy Lmt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "lmt", "day", -10, otEntry);
539                 value1 = OTCreate("Sell Mkt", pAccount, pSymbol, pSecurityType, "sell", 1, Close, "mkt", "day", 0, otStop);
540                 value1 = OTAddActivationRuleTime(otStop, "activate", "absolute", 1459);
541
542             // OSO group for entry and exit tickets
543             oso = new OSOOrderTicket();
544             oso.PrimaryTicket = otEntry;
545             oso.SecondaryTickets.Add(otStop);
546
547             ords = oso.Send(true);
```

9.2 Activate - Relative

This order creates a buy market order 15 seconds after it's placed. Then activates the sell order 15 seconds after that. Note the buy order has 15 seconds on it while the sell order has 30. They both start counting when the order is placed.

```
549         case 41: // Verified
550             // Entry - Single Order - Long with Exit
551             // Relative time activation rule
552             // Activate Buy in 15 seconds, Activate Sell Stop 15 seconds after that
553             // Time starts counting when the order is sent, not when it's filled
554             valuel = OTCreate("Buy Mkt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "mkt", "day", 0, otEntry);
555             valuel = ELF_OTAddActivationRuleTime(otEntry, "activate", "relative", 15);
556                 valuel = OTCreate("Sell Mkt", pAccount, pSymbol, pSecurityType, "sell", 1, Close, "mkt", "day", 0, otStop);
557                 valuel = ELF_OTAddActivationRuleTime(otStop, "activate", "relative", 30);
558
559             // OSO group for entry and exit tickets
560             oso = new OSOOrderTicket();
561             oso.PrimaryTicket = otEntry;
562             oso.SecondaryTickets.Add(otStop);
563
564             ords = oso.Send(true);
565
```

9.3 Deactivate - Absolute

Here I had to create an absolute time outside of the function and then pass it in to the otStop order to test it. ELTime could be substituted for Time + 1, or 1245. The lowest time interval is 1 min for absolute time rules.

```
571         case 42: // Verified
572             // Entry - Single Order - Long with Exit
573             // Time is in ELTime (HHMM, no seconds)
574             // Absolute time deactivation rule
575             valuel = OTCreate("Buy Mkt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "mkt", "day", 0, otEntry);
576                 valuel = OTCreate("Sell Mkt", pAccount, pSymbol, pSecurityType, "sell", 1, Close, "lmt", "day", 10, otStop);
577
578                 // Create an absolute time in the future.  Relative already does this, but this is just to test the functionality
579                 // Adding 1 minute will set the expiration time to the start of the next minute
580                 // If the time is 13:45:22 and 1 minute is added, the expiration time will be 13:46:00
581                 dt = DateTime.Now;
582                 dt.AddMinutes(1);
583                 valuel = OTAddActivationRuleTime(otStop, "deactivate", "absolute", dt.ELTime);
584
585             // OSO group for entry and exit tickets
586             oso = new OSOOrderTicket();
587             oso.PrimaryTicket = otEntry;
588             oso.SecondaryTickets.Add(otStop);
589
590             ords = oso.Send(true);
```

9.4 Deactivate - Relative

This deactivation rules is placed on the entry order. It tries to buy a limit order with a 10 Tick offset. If it doesn't fill within 10 seconds, it cancels the entry order as well as the sell order.

```
592         case 43: // Verified
593             // Entry - Single Order - Long with Exit
594             // Relative time deactivation rule
595             // Buy deactivation rule if order is not filled within 10 seconds
596             // Time starts counting when the order is sent, not when it's filled
597             valuel = OTCreate("Buy Lmt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "lmt", "day", -10, otEntry);
598             valuel = OTAddActivationRuleTime(otEntry, "deactivate", "relative", 10);
599                 valuel = OTCreate("Sell Mkt", pAccount, pSymbol, pSecurityType, "sell", 1, Close, "mkt", "gtc", 0, otStop);
600
601             // OSO group for entry and exit tickets
602             oso = new OSOOrderTicket();
603             oso.PrimaryTicket = otEntry;
604             oso.SecondaryTickets.Add(otStop);
605
606             ords = oso.Send(true);
```

9.5 Price Activation - Single Rule

This price activation rule operates in sort of a stealth mode. It waits for the price to get within 3 ticks of the buy stop order then it activates the buy stop. That way it's not sitting out there in the order book for a market maker to run a stop on. It tries to catch momentum from a rising symbol.

In the following example, the stop market order will become active if the price drops 3 ticks below the price it was issued. If the price rises, before the activation rule triggers, the order will not fill.

```
611         case 44:
612             // Entry - Single Order - Long with Exit
613             // Add price activation rule
614             // Enable the Buy Stop order when the price is 3 ticks below the Close of the Buy Stop
615             // See advanced column of TradeManager for details
616             value1 = OTCreate("Buy StpMkt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "stpmkt", "gtc", 6, otEntry);
617             value1 = OTAddActivationRulePrice(otEntry, pSymbol, Close - 3 * tickSize, "lt", "stt", "and");
618
619             ords = otEntry.Send(true);
```

9.6 Price Activation - Multi Rule

This is a rather complex entry rule that shows how to use and's and or's with other symbols as well as "gt" for > and "lt" for <. The description of how it works is in the code below. This order may have trouble loading due to all the prices and conditions it has. It's more to show how it works than anything.

```
621         case 45:
622             // Entry - Single Order - Long with Exit
623             // Add multiple price activation rules
624             // Enable the Buy Market when the (price is 10 ticks under the Close and $VIX.X > 12) or (TLT < 146.50)
625             // See advanced column of TradeManager for details
626             // Note: Default logical operation is AND, and needs to be set on the first rule if others are to be added
627             // You may have to adjust prices on $VIX and TLT to get the order to fill, but you can see the order in the TradeManger
628             value1 = OTCreate("Buy Mkt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "mkt", "gtc", 0, otEntry);
629             value1 = OTAddActivationRulePrice(otEntry, pSymbol, Close - 10 * tickSize, "lt", "stt", "and");
630             value1 = OTAddActivationRulePrice(otEntry, "$VIX.X", 12, "gt", "stt", "or");
631             value1 = OTAddActivationRulePrice(otEntry, "TLT", 146.50, "le", "stt", "and");
632
633             ords = otEntry.Send(true);
```

10 Bracket Orders

The methods shown here to create bracket orders are the same methods used above to create OCO orders. This means we don't have to work with the many properties of a bracket order and can be consistent in using the OTCreate function for bracket orders.

Below is a buy order with 1 limit and 1 stop. All 3 orders are created using the OTCreate function. The Limit order is assigned the Target.PrimaryTicket. The Stop order is assigned the Protection.PrimaryTicket. This bracket order is then added to an OSO order with the entry ticket and sent to the Trade Server.

All other bracket orders use the same technique.

```
//------------------------------------------------------------------------
//   OCO/OSO - Buy /w 1 stop and 1 target
//------------------------------------------------------------------------
case 20: // Verified
    // Entry - Single Order - Long with Exit - Bracket - 1 Limit & 1 Stop Level
    // Buy Limit offset 10 ticks, Sell Target ParentPlus offset 50 ticks, Sell Stop ParentPlus offset 25 ticks
    valuel = OTCreate("Buy Lmt", pAccount, pSymbol, pSecurityType, "buy", 1, Close, "lmt", "gtc", -10, otEntry);
        valuel = OTCreate("Sell Lmt PP", pAccount, pSymbol, pSecurityType, "sell", 1, Close, "lmt-pp", "gtc", 50, otTarget);
        valuel = OTCreate("Sell StpMkt PP", pAccount, pSymbol, pSecurityType, "sell", 1, Close, "stpmkt-pp", "gtc", -25, otStop);

    // Bracket group for entry and exit tickets
    bot = new BracketOrderTicket();
    bot.Target.PrimaryTicket = otTarget;
    bot.Protection.PrimaryTicket = otStop;

    // OSO group for entry and exit tickets
    oso = new OSOOrderTicket();
    oso.PrimaryTicket = otEntry;
    oso.SecondaryTickets.Add(bot);

    ords = oso.Send(true);
```

11 Short Orders

The short OCO orders have matching ID's with the long orders so they can easily be compared. Only the short orders that made sense to convert from long orders were converted. The same goes for the short bracket orders.

12 Final Thoughts

12.1 Design Considerations - Methods to my Madness

My choice for naming the functions is to keep them grouped together. OT for OrderTicket. The names are the same up to the last part such as, OTActivationRule'Price' or 'Time'. This keeps them in order when I'm looking through the project or opening them in the TDE.

The OTCreate method I grabbed somewhere from the forum, originally had a parameter in it for an Activation rule. I wasn't using activation rules at the time, but I kept having to pass a value for the parameter. I decided to pull this out of the function and create two separate functions that would "decorate" the OrderTicket (not in the true sense of the decorator design pattern). This is how OTActivationRuleTime and OTActivationRulePrice came to be. You'll see this in use in the **LongOCOOrders** method for cases 40 thru 45.

Passing SecurityType to OTCreate. This was a tough one to add. I wanted to create a SymbolAttributesProvider in the function to look up the symbol to determine the SecurityType, but I couldn't consistently get it to load in time to use it. So it was either pass the SAP to the function to get one property, or pass the SecurityType. I figured developers know what they're trading, I'll let them set the value manually.

Global variables - not a fan of them. I try to do most of my work in methods where I can localize the variables to the task at hand. This keeps my global variables to a minimum, which is usually made up of collections, data providers and logging variables.

You may notice that I pass some global variables to a method. Why do that when I can just grab it from the global variable? A couple reasons. 1 - It makes the routines more portable and loosely coupled. 2 - It's usually only for variables that I may want to change. An example is Symbol. If I trade $SPX.X, that's not a tradeable symbol. Besides, the SecurityType for that is Index, but the tradable security is IndexOption. Which brings me back to why SecurityType was passed into OTCreate.

Why didn't I determine + or - for offsets by looking at buy/sell and market/limit settings so all I had to pass was positive values? When I first started this it was helpful to think through what I was doing and I had to know the direction and where the orders were being placed. Also it was in line with how the Trade Builder works. Lastly I didn't like passing in a + value and seeing a - value come out on the PrintLog statements.

12.2 Wrapping up

At this point you know all the OrderTypes you can create using the OTCreate function. You know what each one of them mean. We know when to use Points and when to use Ticks. Prices are in Points, Offsets are in Ticks. We also know how to find the code that generated the order.

We know what to look for in the TradeManager to see how the order was created. We also know how to look at the log to see how the ticket was created using OrderTickets.

12.3 Some final tips

When I first started this app, I used the trade bar to create all the trades, even the simple ones. I'd build the order then submit it. This would put the order in the TradeManager. Then I'd try to replicate it using OrderTickets. Once I was able to get the same results from the OrderTickets I was getting from the Trade Bar, I'd move on to the next one.

If you're having trouble building your own order using OrderTickets, try building it in the Trade Bar first and make it work there. Then try to build it again using OrderTickets.

Sometimes you'll build an order and send it and it won't place the order, but you don't get a notification back as to what happened. These are tough to track down. This usually happens when the Duration for the ticket is wrong for the current session. Build the order at the Trade Bar and look at the valid Durations. Then compare them to the duration you have set on your ticket. Adjust your ticket accordingly.

12.4 Errors & Omissions

If you find a bug or something I missed, let me know, so I can fix it and update it on the forum.

12.5 Thank you

This indicator was made possible by all the Forum Users and TradeStation Engineers that took time to answer my many questions. Thank you all very much.

All the best,

Snaggs