```
#region Using declarations
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media;
using System.Xml.Serialization;
using NinjaTrader.Cbi;
using NinjaTrader.Gui;
using NinjaTrader.Gui.Chart;
using NinjaTrader.Gui.SuperDom;
using NinjaTrader.Gui.Tools;
using NinjaTrader.Data;
using NinjaTrader.NinjaScript;
using NinjaTrader.Core.FloatingPoint;
using NinjaTrader.NinjaScript.DrawingTools;
#endregion

//This namespace holds Indicators in this folder and is required.
Do not change it.
namespace NinjaTrader.NinjaScript.Indicators
{
    public class NLegPullback : Indicator
    {
        enum Side
        {
            Long,
            Short
        }

        private int updown = 0;
        private int pullbackId = 1;
```

```
        private int legCount = 0;
        private EMA ema20;
        private EMA ema50;
        private Swing swing;
        private double highest;
        private double lowest = Double.MaxValue;
        private int legStartBar = 0;
        private double legStartPrice;
        private Side side;

        protected override void OnStateChange()
        {
            if (State == State.SetDefaults)
            {
                Description                                 =
@"Detects n-legs pullbacks either on the long or short side and
visualizes the leg counts and wires them together";
                Name                                        =
"NLegPullback";
                Calculate                                   =
Calculate.OnBarClose;
                IsOverlay                                   =
true;
                DisplayInDataBox                            = true;
                DrawOnPricePanel                            = true;
                DrawHorizontalGridLines                     =
true;
                DrawVerticalGridLines                       =
true;
                PaintPriceMarkers                           =
true;
                ScaleJustification                          =
NinjaTrader.Gui.Chart.ScaleJustification.Right;
                //Disable this property if your indicator requires
custom values that cumulate with each new market data event.
                //See Help Guide for additional information.
                IsSuspendedWhileInactive                    = true;
```

```
                Strict                      = true;
            }
            else if (State == State.DataLoaded)
            {
                this.ema20 = EMA(20);
                this.ema50 = EMA(50);
                this.swing = Swing(5);
            }
        }

        private void endLegLong(String reason)
        {
            if (legStartBar < CurrentBar-1 && legCount>1)
            {
                if (legCount % 2 == 1)
                {
                    Draw.Text(this, "pbe" + pullbackId + "_" +
legCount, "X", 0, Low[0] - TickSize * 2,
ChartControl.Properties.ChartText);
                    Draw.Line(this, "pbl" + pullbackId + "_" +
legCount, true, CurrentBar - legStartBar, legStartPrice, 0, Low[0],
Brushes.Purple, DashStyleHelper.Dot, 2);
                }
                else
                {
                    Draw.Text(this, "pbe" + pullbackId + "_" +
legCount, "X", 0, High[0] + TickSize * 2,
ChartControl.Properties.ChartText);
                    Draw.Line(this, "pbl" + pullbackId + "_" +
legCount, true, CurrentBar - legStartBar, legStartPrice, 0,
High[0], Brushes.Purple, DashStyleHelper.Dot, 2);
                }
            }

            legCount = 0;
            pullbackId++;
        }
```

```
        private void endLegShort(String reason)
        {
            if (legStartBar < CurrentBar - 1 && legCount>1)
            {
                if (legCount % 2 == 0)
                {
                    Draw.Text(this, "pbe" + pullbackId + "_" +
legCount, "X", 0, Low[0] - TickSize * 2,
ChartControl.Properties.ChartText);
                    Draw.Line(this, "pbl" + pullbackId + "_" +
legCount, true, CurrentBar - legStartBar, legStartPrice, 0, Low[0],
Brushes.Purple, DashStyleHelper.Dot, 2);
                }
                else
                {
                    Draw.Text(this, "pbe" + pullbackId + "_" +
legCount, "X", 0, High[0] + TickSize * 2,
ChartControl.Properties.ChartText);
                    Draw.Line(this, "pbl" + pullbackId + "_" +
legCount, true, CurrentBar - legStartBar, legStartPrice, 0,
High[0], Brushes.Purple, DashStyleHelper.Dot, 2);
                }
            }

            legCount = 0;
            pullbackId++;
        }

        protected override void OnBarUpdate()
        {
            if (CurrentBar > 20)
            {
                if (legCount == 0)
                {
                    if (ema20[0] > ema50[0] && Low[0] > ema20[0])
                    {
```

```
                        if (High[0] > swing.SwingHigh[1])
                        {
                            side = Side.Long;
                            updown = -1;
                            int highBar = 0;
                            int lowBar = 0;
                            if (High[0] < High[1])
                            {
                                highBar = 1;
                            }
                            //Draw.Diamond(this, "pbd" +
pullbackId+"_"+legCount, true, Time[highBar], High[highBar] +
TickSize, Brushes.Red);
                            Draw.Text(this, "pbt" + pullbackId +
"_" + legCount, "0", highBar, High[highBar] + TickSize * 2,
ChartControl.Properties.ChartText);

                            highest = High[highBar];
                            lowest = Double.MaxValue;

                            legStartBar = CurrentBar - highBar;
                            legStartPrice = High[highBar];

                            legCount++;

                        }
                    }
                    else if (ema20[0] < ema50[0] && High[0] <
ema20[0])
                    {
                        if (Low[0] < swing.SwingLow[1])
                        {
                            side = Side.Short;
                            updown = 1;
                            int highBar = 0;
                            int lowBar = 0;
                            if (Low[1] < Low[0])
```

```
                                {
                                    lowBar = 0;
                                }
                                //Draw.Diamond(this, "pbd" +
pullbackId+"_"+legCount, true, Time[highBar], High[highBar] +
TickSize, Brushes.Red);
                                Draw.Text(this, "pbt" + pullbackId +
"_" + legCount, "0", lowBar, Low[lowBar] - TickSize * 2,
ChartControl.Properties.ChartText);

                                highest = 0;
                                lowest = Low[lowBar];

                                legStartBar = CurrentBar - lowBar;
                                legStartPrice = Low[lowBar];

                                legCount++;
                                                }
                        }
                    }
                    else
                    {

                        if (side == Side.Long)
                        {
                            if (High[0] > highest)
                            {
                                endLegLong("X0");
                                return;
                            }
                            if (updown == -1 && High[0] > High[1])
                            {

                                int lowBar = 1;

                                if (legStartBar < CurrentBar - lowBar)
                                {
```

```
                                    if (Low[0]>lowest && Strict)
                                    {
                                        endLegLong("X1");
                                        return;
                                    }

                                    updown = 1;

                                    //Draw.Diamond(this, "pbd" +
pullbackId + "_" + legCount, true, Time[lowBar], Low[lowBar] -
TickSize, Brushes.Red);
                                    Draw.Text(this, "pbt" + pullbackId
+ "_" + legCount, "" + legCount, lowBar, Low[lowBar] - TickSize *
2, ChartControl.Properties.ChartText);
                                    lowest = Low[lowBar];

                                    Draw.Line(this, "pbl" + pullbackId
+ "_" + legCount, false, CurrentBar - legStartBar, legStartPrice,
lowBar, Low[lowBar], Brushes.Purple, DashStyleHelper.Dot, 2);
                                    legStartBar = CurrentBar - lowBar;
                                    legStartPrice = Low[lowBar];

                                    legCount++;
                                }
                            }
                            else if (updown == 1 && Low[0] < Low[1])
                            {
                                updown = -1;
                                int highBar = 1;
                                //Draw.Diamond(this, "pbd" + pullbackId
+ "_" + legCount, true, Time[highBar], High[highBar] + TickSize,
Brushes.Red);
                                Draw.Text(this, "pbt" + pullbackId +
"_" + legCount, "" + legCount, highBar, High[highBar] + TickSize *
2, ChartControl.Properties.ChartText);
                                highest = High[highBar];
```

```
                                Draw.Line(this, "pbl" + pullbackId +
"_" + legCount, false, CurrentBar - legStartBar, legStartPrice,
highBar, High[highBar], Brushes.Purple, DashStyleHelper.Dot, 2);
                                legStartBar = CurrentBar - highBar;
                                legStartPrice = High[highBar];

                                legCount++;
                            }

                        }
                        else if (side == Side.Short)
                        {
                            if (Low[0] < lowest)
                            {
                                endLegShort("X0");
                                return;
                            }
                            if (updown == 1 && Low[0] < Low[1])
                            {

                                int highBar = 1;

                                if (legStartBar < CurrentBar - highBar)
                                {
                                    updown = -1;

                                    if (High[0] < highest && Strict)
                                    {
                                        endLegShort("X1");
                                        return;
                                    }

                                    //Draw.Diamond(this, "pbd" +
pullbackId + "_" + legCount, true, Time[lowBar], Low[lowBar] -
TickSize, Brushes.Red);
```

```
                                Draw.Text(this, "pbt" + pullbackId
+ "_" + legCount, "" + legCount, highBar, High[highBar] + TickSize
* 2, ChartControl.Properties.ChartText);
                                highest = High[highBar];

                                Draw.Line(this, "pbl" + pullbackId
+ "_" + legCount, false, CurrentBar - legStartBar, legStartPrice,
highBar, High[highBar], Brushes.Purple, DashStyleHelper.Dot, 2);
                                legStartBar = CurrentBar - highBar;
                                legStartPrice = High[highBar];

                                legCount++;
                            }
                        }
                        else if (updown == -1 && High[0] > High[1])
                        {
                            updown = 1;
                            int lowBar = 1;
                            //Draw.Diamond(this, "pbd" + pullbackId
+ "_" + legCount, true, Time[highBar], High[highBar] + TickSize,
Brushes.Red);
                            Draw.Text(this, "pbt" + pullbackId +
"_" + legCount, "" + legCount, lowBar, Low[lowBar] - TickSize * 2,
ChartControl.Properties.ChartText);
                            lowest = Low[lowBar];

                            Draw.Line(this, "pbl" + pullbackId +
"_" + legCount, false, CurrentBar - legStartBar, legStartPrice,
lowBar, Low[lowBar], Brushes.Purple, DashStyleHelper.Dot, 2);
                            legStartBar = CurrentBar - lowBar;
                            legStartPrice = Low[lowBar];

                            legCount++;
                        }

                    }
                }
```

```
            }

        }

        #region Properties
        [NinjaScriptProperty]
        [Display(Name="Strict", Description="Terminate pullback as
soon as low/high pivot is not adhering the Z-shape", Order=1,
GroupName="Parameters")]
        public bool Strict
        { get; set; }
        #endregion

    }
}

#region NinjaScript generated code. Neither change nor remove.

namespace NinjaTrader.NinjaScript.Indicators
{
    public partial class Indicator :
NinjaTrader.Gui.NinjaScript.IndicatorRenderBase
    {
        private NLegPullback[] cacheNLegPullback;
        public NLegPullback NLegPullback(bool strict)
        {
            return NLegPullback(Input, strict);
        }

        public NLegPullback NLegPullback(ISeries<double> input,
bool strict)
        {
            if (cacheNLegPullback != null)
                for (int idx = 0; idx < cacheNLegPullback.Length;
idx++)
```

```
                    if (cacheNLegPullback[idx] != null &&
cacheNLegPullback[idx].Strict == strict &&
cacheNLegPullback[idx].EqualsInput(input))
                        return cacheNLegPullback[idx];
            return CacheIndicator<NLegPullback>(new NLegPullback(){
Strict = strict }, input, ref cacheNLegPullback);
        }
    }
}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns
{
    public partial class MarketAnalyzerColumn :
MarketAnalyzerColumnBase
    {
        public Indicators.NLegPullback NLegPullback(bool strict)
        {
            return indicator.NLegPullback(Input, strict);
        }

        public Indicators.NLegPullback NLegPullback(ISeries<double>
input , bool strict)
        {
            return indicator.NLegPullback(input, strict);
        }
    }
}

namespace NinjaTrader.NinjaScript.Strategies
{
    public partial class Strategy :
NinjaTrader.Gui.NinjaScript.StrategyRenderBase
    {
        public Indicators.NLegPullback NLegPullback(bool strict)
        {
            return indicator.NLegPullback(Input, strict);
        }
```

```
        public Indicators.NLegPullback NLegPullback(ISeries<double>
input , bool strict)
        {
            return indicator.NLegPullback(input, strict);
        }
    }
}

#endregion
```