```
// Vers 0.1 - translated by Gumphrie
// Vers 0.2 - removed superfluous MACD dot plotted on the zero line for the current bar.


#region Using declarations
using System;
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.ComponentModel;
using System.Xml.Serialization;
using NinjaTrader.Data;
using NinjaTrader.Gui.Chart;
using NinjaTrader.Gui.Design;
#endregion

// This namespace holds all indicators and is required. Do not change it.
namespace NinjaTrader.Indicator
{
/// <summary>
/// MACD BB Lines
/// </summary>
[Description("MACD BB Lines.")]
[Gui.Design.DisplayName("MACD BB Lines")]
public class MACDBBLines : Indicator
{
#region Variables
private intfast= 12;
private intslow= 26;
private intsmooth= 5;
private int stDv = 1;
private int length = 10;
private SolidBrush        BandFillBrush = new SolidBrush(Color.LightBlue);
private SolidBrush        UpLineBrush   = new SolidBrush(Color.Blue);
private SolidBrush        DownLineBrush = new SolidBrush(Color.Red);
private SolidBrush        UpBrush   = new SolidBrush(Color.LimeGreen);
private SolidBrush        DownBrush   = new SolidBrush(Color.Red);
private SolidBrush        AlertBrush   = new SolidBrush(Color.Yellow);

private int               currentBar    = 0;
private int               fillAlpha     = 64;
private int               zeroLineWidth = 1;
private bool              showZeroLineCross=true;
private DataSeries bbMacd;
private DataSeries dotSeries;
#endregion

/// <summary>
/// This method is used to configure the indicator and is called once before any bar data is loaded.
/// </summary>
```

```
protected override void Initialize()
{
Add(new Plot(new Pen(Color.Red, 1), "BB_UpperBand"));
Add(new Plot(new Pen(Color.Blue, 1), "BB_LowerBand"));

bbMacd    = new DataSeries(this);
dotSeries = new DataSeries(this);

DrawOnPricePanel=false;
}

/// <summary>
/// Calculates the indicator value(s) at the current index.
/// </summary>
protected override void OnBarUpdate()
{

bbMacd.Set(MACD(Input, fast, slow, smooth)[0]);

double Avg = EMA(bbMacd,length)[0];
double SDev = StdDev(bbMacd,length)[0];
double upperBand=Avg+(stDv*SDev);
double lowerBand=Avg-(stDv*SDev);

BB_UpperBand.Set(upperBand);
  BB_LowerBand.Set(lowerBand);
dotSeries.Set(bbMacd[0]);
currentBar=CurrentBar;
}



public override void GetMinMaxValues(ChartControl chartControl, ref double min, ref double max)
{
base.GetMinMaxValues(chartControl, ref min, ref max);

int bars = base.ChartControl.BarsPainted;
        int barPaintWidth = base.ChartControl.ChartStyle.GetBarPaintWidth(base.ChartControl.BarWi
Exception caughtException;



        while (bars >= 0)
        {
int index = ((base.ChartControl.LastBarPainted - base.ChartControl.BarsPainted) + 1) + bars;
            if (base.ChartControl.ShowBarsRequired || ((index - base.Displacement) >= base.BarsRequi
            {
try
{
int x1 = (((base.ChartControl.CanvasRight - base.ChartControl.BarMarginRight) - (barPaintWidth / 2))
double macdVal =  dotSeries.Get(index);
```

```
          if ((!double.IsNaN(macdVal)) && (index>0))
{
if (macdVal>max) max=macdVal;
if (macdVal<min) min=macdVal;
}
}
catch (Exception exception) {caughtException=exception;}
}
bars--;
}

}


public override void Plot(Graphics graphics, Rectangle bounds, double min, double max)
{
// Default plotting in base class.
base.Plot(graphics, bounds, min, max);

if (base.Bars == null) return;

      Exception caughtException;

      int index = -1;
byte bRed = (byte)~(base.ChartControl.BackColor.R);
         byte bGreen = (byte)~(base.ChartControl.BackColor.G);
byte bBlue = (byte)~(base.ChartControl.BackColor.B);


Color borderColor = Color.FromArgb(bRed,bGreen,bBlue);



int bars = base.ChartControl.BarsPainted;
         int barPaintWidth = base.ChartControl.ChartStyle.GetBarPaintWidth(base.ChartControl.BarWi
bool lineUp=false;
int LastX=ChartControl.CanvasRight;
int y = 0;

         while (bars >= 0)
         {
index = ((base.ChartControl.LastBarPainted - base.ChartControl.BarsPainted) + 1) + bars;
            if (base.ChartControl.ShowBarsRequired || ((index - base.Displacement) >= base.BarsRequi
            {
try
{
int x1 = (((base.ChartControl.CanvasRight - base.ChartControl.BarMarginRight) - (barPaintWidth / 2))
double macdVal =  dotSeries.Get(index);
```

```
        if ((!double.IsNaN(macdVal)) && (index>0))
{
double prevMacdVal =  dotSeries.Get(index-1);

y = (bounds.Y + bounds.Height) - ((int) (((0 - min) / ChartControl.MaxMinusMin(max, min)) * bounds.

if ((prevMacdVal<=0 && macdVal>0) || (prevMacdVal>=0 && macdVal<0))
{
Pen linePen;

if  (prevMacdVal<=0 && macdVal>0)
{
linePen = new Pen(UpLineBrush.Color,zeroLineWidth);
lineUp = true;
}
else
{
linePen = new Pen(DownLineBrush.Color,zeroLineWidth);
lineUp = false;
}

graphics.DrawLine(linePen,ChartControl.CanvasLeft,(int)y,LastX,(int)y);
                        LastX=x1;

if (showZeroLineCross)
{
SmoothingMode smoothingMode = graphics.SmoothingMode;
   graphics.SmoothingMode = SmoothingMode.AntiAlias;

Pen bigpen = new Pen(AlertBrush.Color, 20);
bigpen.DashStyle = DashStyle.Dot;
bigpen.DashCap = DashCap.Round;

graphics.DrawPie(bigpen,x1-5,y-2,5,5,0,360);

graphics.SmoothingMode = smoothingMode;
}
}
}
}
catch (Exception exception) {caughtException=exception;}
}
bars--;
}

Pen lastLinePen;

if  (lineUp)
{
lastLinePen = new Pen(DownLineBrush.Color,zeroLineWidth);
```

```
}
else
{
lastLinePen = new Pen(UpLineBrush.Color,zeroLineWidth);
}

graphics.DrawLine(lastLinePen,ChartControl.CanvasLeft,(int)y,LastX,(int)y);


bars = base.ChartControl.BarsPainted;

        while (bars >= 0)
        {
index = ((base.ChartControl.LastBarPainted - base.ChartControl.BarsPainted) + 1) + bars;
            if (base.ChartControl.ShowBarsRequired || ((index - base.Displacement) >= base.BarsRequi
            {
try
{
int x1 = (((base.ChartControl.CanvasRight - base.ChartControl.BarMarginRight) - (barPaintWidth / 2))

                double upperVal = BB_UpperBand.Get(index);
double lowerVal = BB_LowerBand.Get(index);
double macdVal =  dotSeries.Get(index);

if ((!double.IsNaN(upperVal)) && (!double.IsNaN(lowerVal)) && (index>0))
                {
double prevUpperVal = BB_UpperBand.Get(index-1);
   double prevLowerVal = BB_LowerBand.Get(index-1);
int y1 = (bounds.Y + bounds.Height) - ((int) (((upperVal - min) / ChartControl.MaxMinusMin(max, min
                    int x2 = x1-base.ChartControl.BarSpace;
int y2 = (bounds.Y + bounds.Height) - ((int) (((lowerVal - min) / ChartControl.MaxMinusMin(max, min
                    int y3 = (bounds.Y + bounds.Height) - ((int) (((prevLowerVal - min) / ChartControl.Ma
                    int y4 = (bounds.Y + bounds.Height) - ((int) (((prevUpperVal - min) / ChartControl.Ma
Point[] points = new Point[] { new Point(x1, y1), new Point(x1, y2), new Point(x2, y3), new Point(x2, y

graphics.FillPolygon(new SolidBrush(Color.FromArgb(fillAlpha, BandFillBrush.Color)),points);
}
if ((!double.IsNaN(macdVal)) && (index>0) && (index<=currentBar))
{
Color dotColor = Color.Transparent;
double prevMacdVal =  dotSeries.Get(index-1);

y = (bounds.Y + bounds.Height) - ((int) (((macdVal - min) / ChartControl.MaxMinusMin(max, min)) * b


if (macdVal>prevMacdVal) dotColor = UpBrush.Color;
else dotColor = DownBrush.Color;

SmoothingMode smoothingMode = graphics.SmoothingMode;
   graphics.SmoothingMode = SmoothingMode.AntiAlias;
```

```
Pen pen = new Pen(borderColor, 7);
pen.DashStyle = DashStyle.Dot;
pen.DashCap = DashCap.Round;

graphics.DrawPie(pen,x1-1,y-1,2,2,0,360);

pen = new Pen(dotColor, 5);
pen.DashStyle = DashStyle.Dot;
pen.DashCap = DashCap.Round;

graphics.DrawRectangle(pen,x1-2,y-2,2,2);

graphics.SmoothingMode = smoothingMode;
}
}
catch (Exception exception) {caughtException=exception;}
}
bars--;
}

}

#region Properties
/// <summary>
/// </summary>
[Browsable(false)]
[XmlIgnore()]
public DataSeries BB_UpperBand
{
get { return Values[0]; }
}

      /// <summary>
/// </summary>
[Browsable(false)]
[XmlIgnore()]
public DataSeries BB_LowerBand
{
get { return Values[1]; }
}


/// <summary>
/// </summary>
[Description("Number of bars for fast EMA")]
[Category("Parameters")]
public int Fast
{
get { return fast; }
```

```
set { fast = Math.Max(1, value); }
}

/// <summary>
/// </summary>
[Description("Number of bars for slow EMA")]
[Category("Parameters")]
public int Slow
{
get { return slow; }
set { slow = Math.Max(1, value); }
}

/// <summary>
/// </summary>
[Description("Number of bars for smoothing")]
[Category("Parameters")]
public int Smooth
{
get { return smooth; }
set { smooth = Math.Max(1, value); }
}

/// <summary>
/// </summary>
[Description("Standard Deviation Factor")]
[Category("Parameters")]
public int StDev
{
get { return stDv; }
set { stDv = Math.Max(1, value); }
}

/// <summary>
/// </summary>
[Description("Period")]
[Category("Parameters")]
public int Period
{
get { return length; }
set { length = Math.Max(1, value); }
}

[Browsable(false)]
   public string BandFillColorSerialize
   {
      get { return SerializableColor.ToString(this.BandFillColor); }
      set { this.BandFillColor = SerializableColor.FromString(value); }
   }
```

```
[Description("Default Colour to Fill Bollinger Bands"), XmlIgnore, VisualizationOnly]
[Category("Plots")]
[NinjaTrader.Gui.Design.DisplayName("BB fill color")]
public Color BandFillColor
      {
get { return this.BandFillBrush.Color; }
      set { this.BandFillBrush = new SolidBrush(value); }
      }

[Browsable(false)]
   public string BandFillAlphaSerialize
   {
      get { return this.BandFillAlpha.ToString(); }
      set { this.BandFillAlpha = Convert.ToInt32(value); }
   }

[Description("Alpha setting for band fill colour. Range is 0 to 255.")]
      [Category("Plots")]
[NinjaTrader.Gui.Design.DisplayName("BB fill alpha")]
      public int BandFillAlpha
      {
         get { return fillAlpha; }
         set { fillAlpha = Math.Max(0, Math.Min(255,value)); }
      }

[Browsable(false)]
   public string UpColorSerialize
   {
      get { return SerializableColor.ToString(this.UpColor); }
      set { this.UpColor = SerializableColor.FromString(value); }
   }

[Description("Default Colour for Rising MACD"), XmlIgnore, VisualizationOnly]
[Category("Plots")]
[NinjaTrader.Gui.Design.DisplayName("MACD rising color")]
public Color UpColor
      {
get { return this.UpBrush.Color; }
      set { this.UpBrush = new SolidBrush(value); }
      }

[Browsable(false)]
   public string DownColorSerialize
   {
      get { return SerializableColor.ToString(this.DownColor); }
      set { this.DownColor = SerializableColor.FromString(value); }
   }

[Description("Default Colour for Falling MACD"), XmlIgnore, VisualizationOnly]
[Category("Plots")]
```

```
[NinjaTrader.Gui.Design.DisplayName("MACD falling color")]
public Color DownColor
      {
get { return this.DownBrush.Color; }
      set { this.DownBrush = new SolidBrush(value); }
      }

[Browsable(false)]
   public string UpLineColorSerialize
   {
      get { return SerializableColor.ToString(this.UpLineColor); }
      set { this.UpLineColor = SerializableColor.FromString(value); }
   }

[Description("Default Zero Line Colour for Rising MACD"), XmlIgnore, VisualizationOnly]
[Category("Plots")]
[NinjaTrader.Gui.Design.DisplayName("Zero line rising color")]
public Color UpLineColor
      {
get { return this.UpLineBrush.Color; }
      set { this.UpLineBrush = new SolidBrush(value); }
      }

[Browsable(false)]
   public string DownLineColorSerialize
   {
      get { return SerializableColor.ToString(this.DownLineColor); }
      set { this.DownLineColor = SerializableColor.FromString(value); }
   }

[Description("Default Zero Line Colour for Falling MACD"), XmlIgnore, VisualizationOnly]
[Category("Plots")]
[NinjaTrader.Gui.Design.DisplayName("Zero line falling color")]
public Color DownLineColor
      {
get { return this.DownLineBrush.Color; }
      set { this.DownLineBrush = new SolidBrush(value); }
      }

[Browsable(false)]
   public string ZeroLineWidthSerialize
   {
      get { return this.ZeroLineWidth.ToString(); }
      set { this.ZeroLineWidth = Convert.ToInt32(value); }
   }

[Description("Default Zero Line Width")]
      [Category("Plots")]
[NinjaTrader.Gui.Design.DisplayName("Zero line width")]
      public int ZeroLineWidth
```

```
        {
            get { return zeroLineWidth; }
            set { zeroLineWidth = Math.Max(0, value); }
        }

[Description("Whether to highlight Zero Line crosses")]
[Category("Plots")]
[NinjaTrader.Gui.Design.DisplayName("Zero line cross alerts")]
public bool ShowZeroLineCross
        {
            get { return showZeroLineCross; }
set { showZeroLineCross = value; }
        }


[Browsable(false)]
    public string AlertColorSerialize
    {
        get { return SerializableColor.ToString(this.AlertColor); }
        set { this.AlertColor = SerializableColor.FromString(value); }
    }

[Description("Default Colour for Zero Line Cross"), XmlIgnore, VisualizationOnly]
[Category("Plots")]
[NinjaTrader.Gui.Design.DisplayName("Zero line cross alert color")]
public Color AlertColor
        {
get { return this.AlertBrush.Color; }
        set { this.AlertBrush = new SolidBrush(value); }
        }

#endregion
}
}

#region NinjaScript generated code. Neither change nor remove.
// This namespace holds all indicators and is required. Do not change it.
namespace NinjaTrader.Indicator
{
    public partial class Indicator : IndicatorBase
    {
        private MACDBBLines[] cacheMACDBBLines = null;

        private static MACDBBLines checkMACDBBLines = new MACDBBLines();

        /// <summary>
        /// MACD BB Lines.
        /// </summary>
        /// <returns></returns>
        public MACDBBLines MACDBBLines(int fast, int period, int slow, int smooth, int stDev)
```

```
    {
        return MACDBBLines(Input, fast, period, slow, smooth, stDev);
    }

    /// <summary>
    /// MACD BB Lines.
    /// </summary>
    /// <returns></returns>
    public MACDBBLines MACDBBLines(Data.IDataSeries input, int fast, int period, int slow, int smoo
    {
        if (cacheMACDBBLines != null)
            for (int idx = 0; idx < cacheMACDBBLines.Length; idx++)
                if (cacheMACDBBLines[idx].Fast == fast && cacheMACDBBLines[idx].Period == period &&
                    return cacheMACDBBLines[idx];

        lock (checkMACDBBLines)
        {
            checkMACDBBLines.Fast = fast;
            fast = checkMACDBBLines.Fast;
            checkMACDBBLines.Period = period;
            period = checkMACDBBLines.Period;
            checkMACDBBLines.Slow = slow;
            slow = checkMACDBBLines.Slow;
            checkMACDBBLines.Smooth = smooth;
            smooth = checkMACDBBLines.Smooth;
            checkMACDBBLines.StDev = stDev;
            stDev = checkMACDBBLines.StDev;

            if (cacheMACDBBLines != null)
                for (int idx = 0; idx < cacheMACDBBLines.Length; idx++)
                    if (cacheMACDBBLines[idx].Fast == fast && cacheMACDBBLines[idx].Period == period &
                        return cacheMACDBBLines[idx];

            MACDBBLines indicator = new MACDBBLines();
            indicator.BarsRrequired = BarsRequired;
            indicator.CalculateOnBarClose = CalculateOnBarClose;
#if NT7
            indicator.ForceMaximumBarsLookBack256 = ForceMaximumBarsLookBack256;
            indicator.MaximumBarsLookBack = MaximumBarsLookBack;
#endif
            indicator.Input = input;
            indicator.Fast = fast;
            indicator.Period = period;
            indicator.Slow = slow;
            indicator.Smooth = smooth;
            indicator.StDev = stDev;
            Indicators.Add(indicator);
            indicator.SetUp();

            MACDBBLines[] tmp = new MACDBBLines[cacheMACDBBLines == null ? 1 : cacheMACDBBLi
```

```
            if (cacheMACDBBLines != null)
                cacheMACDBBLines.CopyTo(tmp, 0);
            tmp[tmp.Length - 1] = indicator;
            cacheMACDBBLines = tmp;
            return indicator;
        }
    }
  }
}

// This namespace holds all market analyzer column definitions and is required. Do not change it.
namespace NinjaTrader.MarketAnalyzer
{
  public partial class Column : ColumnBase
  {
    /// <summary>
    /// MACD BB Lines.
    /// </summary>
    /// <returns></returns>
    [Gui.Design.WizardCondition("Indicator")]
    public Indicator.MACDBBLines MACDBBLines(int fast, int period, int slow, int smooth, int stDev)
    {
      return _indicator.MACDBBLines(Input, fast, period, slow, smooth, stDev);
    }

    /// <summary>
    /// MACD BB Lines.
    /// </summary>
    /// <returns></returns>
    public Indicator.MACDBBLines MACDBBLines(Data.IDataSeries input, int fast, int period, int slow
    {
      return _indicator.MACDBBLines(input, fast, period, slow, smooth, stDev);
    }
  }
}

// This namespace holds all strategies and is required. Do not change it.
namespace NinjaTrader.Strategy
{
  public partial class Strategy : StrategyBase
  {
    /// <summary>
    /// MACD BB Lines.
    /// </summary>
    /// <returns></returns>
    [Gui.Design.WizardCondition("Indicator")]
    public Indicator.MACDBBLines MACDBBLines(int fast, int period, int slow, int smooth, int stDev)
    {
      return _indicator.MACDBBLines(Input, fast, period, slow, smooth, stDev);
    }
```

```
        /// <summary>
        /// MACD BB Lines.
        /// </summary>
        /// <returns></returns>
        public Indicator.MACDBBLines MACDBBLines(Data.IDataSeries input, int fast, int period, int slow
        {
            if (InInitialize && input == null)
                throw new ArgumentException("You only can access an indicator with the default input/bar

            return _indicator.MACDBBLines(input, fast, period, slow, smooth, stDev);
        }
    }
}
#endregion
```

dth);

red))

- ((base.ChartControl.BarsPainted - 1) * base.ChartControl.BarSpace)) + (bars * base.ChartControl.Bar

dth);

red))

- ((base.ChartControl.BarsPainted - 1) * base.ChartControl.BarSpace)) + (bars * base.ChartControl.Bar

Height));

red))

- ((base.ChartControl.BarsPainted - 1) * base.ChartControl.BarSpace)) + (bars * base.ChartControl.Bar

)) * bounds.Height));

)) * bounds.Height));
xMinusMin(max, min)) * bounds.Height));
xMinusMin(max, min)) * bounds.Height));
'4) };

ounds.Height));

ɔth, int stDev)

cacheMACDBBLines[idx].Slow == slow && cacheMACDBBLines[idx].Smooth == smooth && cacheMA(

:& cacheMACDBBLines[idx].Slow == slow && cacheMACDBBLines[idx].Smooth == smooth && cacheM

nes.Length + 1];

, int smooth, int stDev)

, int smooth, int stDev)

series from within the 'Initialize()' method");

Space);

-Space);

Space);

CDBBLines[idx].StDev == stDev && cacheMACDBBLines[idx].EqualsInput(input))

IACDBBLines[idx].StDev == stDev && cacheMACDBBLines[idx].EqualsInput(input))