AccessRights

Summary

cBots and indicators access rights

Syntax

```
public sealed enum AccessRights
```

Members

Name	Type	Summary
FileSystem	Field	Access to file system.
FullAccess	Field	The unlimited access rights.
Internet	Field	Access to Internet or other networks.
None	Field	Algorithm doesn’t require any access rights.
Registry	Field	Access to windows registry.

None

Summary

Algorithm doesn’t require any access rights.

Syntax

```
AccessRights.None
```

FileSystem

Summary

Access to file system.

Syntax

```
AccessRights.FileSystem
```

Internet

Summary

Access to Internet or other networks.

Syntax

```
AccessRights.Internet
```

Registry

Summary

Access to windows registry.

Syntax

```
AccessRights.Registry
```

FullAccess

Summary

The unlimited access rights.

Syntax

```
AccessRights.FullAccess
```

Chart

Summary

The Chart Interface.

Syntax

```
public interface Chart : ChartArea
```

Members

Name	Type	Summary
BarsTotal	Property	Gets the total number of the bars on the chart.
ChartType	Property	Gets or sets the type of the chart - Bar, Candlesticks, Line or Dots chart.
ChartTypeChanged	Event	Occurs when the chart type changes.
ColorsChanged	Event	Occurs when the chart color settings change.
ColorSettings	Property	Gets the chart color settings.
DisplaySettings	Property	Gets the chart display settings.
DisplaySettingsChanged	Event	Occurs when one or several charts display settings change.
FirstVisibleBarIndex	Property	Gets the index of the first visible bar on the chart.
IndicatorAreaAdded	Event	Occurs when the indicator area is added.
IndicatorAreaRemoved	Event	Occurs when the indicator area is removed.
IndicatorAreas	Property	Gets the read only list of the indicator areas.
IsScrollingEnabled	Property	Gets or sets the value indicating whether the scrolling is enabled or disabled for the chart. If disabled, then the chart is not affected by scrolling, dragging, scaling, or pressing any keyboard keys, but is still affected by resizing, zooming, and API calls for changing X or Y-axis positions on the chart.
LastVisibleBarIndex	Property	Gets the index of the last visible bar on the chart.
MarketSeries	Property	Gets the the chart market data such as Open, High, Low, Close, Median, Typical, and WeightedClose price series, as well as OpenTime for the symbol, SymbolCode, TickVolume, and TimeFrame.
MaxVisibleBars	Property	Gets the maximum number of the visible bars on the chart.
ScrollXBy	Method	Scrolls the chart by the X-axis for the specified number of bars.
ScrollXTo	Method	Scrolls the chart by the X-axis to the bar with the specified index.
Symbol	Property	Gets the chart symbol.
TimeFrame	Property	Gets the time frame of the chart from 1 minute to 1 month.
Zoom	Property	Gets or sets the zoom option from 0 to 5.

ZoomChanged	Event	Occurs when the chart zoom options change.

IndicatorAreas

Summary

Gets the read only list of the indicator areas.

Syntax

```
public IReadonlyList IndicatorAreas{ get; }
```

DisplaySettings

Summary

Gets the chart display settings.

Syntax

```
public ChartDisplaySettings DisplaySettings{ get; }
```

ColorSettings

Summary

Gets the chart color settings.

Syntax

```
public ChartColorSettings ColorSettings{ get; }
```

ChartType

Summary

Gets or sets the type of the chart - Bar, Candlesticks, Line or Dots chart.

Syntax

```
public ChartType ChartType{ get; set; }
```

Zoom

Summary

Gets or sets the zoom option from 0 to 5.

Syntax

```
public int Zoom{ get; set; }
```

FirstVisibleBarIndex

Summary

Gets the index of the first visible bar on the chart.

Syntax

```
public int FirstVisibleBarIndex{ get; }
```

LastVisibleBarIndex

Summary

Gets the index of the last visible bar on the chart.

Syntax

```
```

```
public int LastVisibleBarIndex{ get; }
```

MaxVisibleBars

Summary

Gets the maximum number of the visible bars on the chart.

Syntax

```
public int MaxVisibleBars{ get; }
```

BarsTotal

Summary

Gets the total number of the bars on the chart.

Syntax

```
public int BarsTotal{ get; }
```

MarketSeries

Summary

Gets the the chart market data such as Open, High, Low, Close, Median, Typical, and WeightedClose price series, as well as OpenTime for the symbol, SymbolCode, TickVolume, and TimeFrame.

Syntax

```
public MarketSeries MarketSeries{ get; }
```

TimeFrame

Summary

Gets the time frame of the chart from 1 minute to 1 month.

Syntax

```
public TimeFrame TimeFrame{ get; }
```

Symbol

Summary

Gets the chart symbol.

Syntax

```
public Symbol Symbol{ get; }
```

IsScrollingEnabled

Summary

Gets or sets the value indicating whether the scrolling is enabled or disabled for the chart. If disabled, then the chart is not affected by scrolling, dragging, scaling, or pressing any keyboard keys, but is still affected by resizing, zooming, and API calls for changing X or Y-axis positions on the chart.

Syntax

```
public bool IsScrollingEnabled{ get; set; }
```

ScrollXBy

Summary

Scrolls the chart by the X-axis for the specified number of bars.

Syntax

```
public void ScrollXBy(int bars)
```

Parameters

Name	Description

ScrollXTo

Summary

Scrolls the chart by the X-axis to the bar with the specified index.

Syntax

```
public void ScrollXTo(int barIndex)
```

```
public void ScrollXTo(DateTime time)
```

Parameters

Name	Description

ScrollXTo

Summary

Scrolls the chart by the X-axis to the specified date time.

Syntax

```
public void ScrollXTo(int barIndex)
```

```
public void ScrollXTo(DateTime time)
```

Parameters

Name	Description

DisplaySettingsChanged

Summary

Occurs when one or several charts display settings change.

Syntax

```
public event Action DisplaySettingsChanged
```

ColorsChanged

Summary

Occurs when the chart color settings change.

Syntax

```
public event Action ColorsChanged
```

ChartTypeChanged

Summary

Occurs when the chart type changes.

Syntax

```
public event Action ChartTypeChanged
```

ZoomChanged

Summary

Occurs when the chart zoom options change.

Syntax

```
public event Action ZoomChanged
```

IndicatorAreaAdded

Summary

Occurs when the indicator area is added.

Syntax

```
public event Action IndicatorAreaAdded
```

IndicatorAreaRemoved

Summary

Occurs when the indicator area is removed.

Syntax

```
public event Action IndicatorAreaRemoved
```

ChartAndrewsPitchfork

Summary

Represents the Andrew's Pitchfork chart object. A tool that helps to identify possible support and resistance levels with the three parallel lines.

Syntax

```
public interface ChartAndrewsPitchfork : ChartObject
```

Members

Name	Type	Summary
Color	Property	Gets or sets the chart object lines color.
LineStyle	Property	Gets or sets the chart object lines style.
Thickness	Property	Gets or sets the chart object lines thickness.
Time1	Property	Gets or sets the time value for the Andrew's Pitchfork point 1.
Time2	Property	Gets or sets the time value for the Andrew's Pitchfork point 2.
Time3	Property	Gets or sets the time value for the Andrew's Pitchfork point 3.
Y1	Property	Gets or sets the Y-axis value for the Andrew's Pitchfork point 1.
Y2	Property	Gets or sets the Y-axis value for the Andrew's Pitchfork point 2.
Y3	Property	Gets or sets the Y-axis value for the Andrew's Pitchfork point 3.

Time1

Summary

Gets or sets the time value for the Andrew's Pitchfork point 1.

Syntax

```
public DateTime Time1{ get; set; }
```

Time2

Summary

Gets or sets the time value for the Andrew's Pitchfork point 2.

Syntax

```
public DateTime Time2{ get; set; }
```

Time3

Summary

Gets or sets the time value for the Andrew's Pitchfork point 3.

Syntax

```
public DateTime Time3{ get; set; }
```

Y1

Summary

Gets or sets the Y-axis value for the Andrew's Pitchfork point 1.

Syntax

```
public double Y1{ get; set; }
```

Y2

Summary

Gets or sets the Y-axis value for the Andrew's Pitchfork point 2.

Syntax

```
public double Y2{ get; set; }
```

Y3

Summary

Gets or sets the Y-axis value for the Andrew's Pitchfork point 3.

Syntax

```
public double Y3{ get; set; }
```

Thickness

Summary

Gets or sets the chart object lines thickness.

Syntax

```
public int Thickness{ get; set; }
```

Color

Summary

Gets or sets the chart object lines color.

Syntax

```
public Color Color{ get; set; }
```

LineStyle

Summary

Gets or sets the chart object lines style.

Syntax

```
public LineStyle LineStyle{ get; set; }
```

ChartArea

Summary

The Chart Area Interface.

Syntax

```
public interface ChartArea
```

Members

Name	Type	Summary
BottomY	Property	Gets the lowest visible Y-axis value.
Drag	Event	Occurs when dragging a chart area.
DragEnd	Event	Occurs when mouse button is released while dragging a chart area or a chart area loses mouse capture.
DragStart	Event	Occurs when MouseDown event is happening on a chart area and a mouse is captured for dragging.
DrawAndrewsPitchfork	Method	Draws an Andrew's pitchfork.
DrawEllipse	Method	Draws an ellipse.
DrawEquidistantChannel	Method	Draws an equidistant channel.
DrawFibonacciExpansion	Method	Draws a Fibonacci expansion.
DrawFibonacciFan	Method	Draws a Fibonacci fan.
DrawFibonacciRetracement	Method	Draws a Fibonacci retracement.
DrawHorizontalLine	Method	Draws a horizontal line.
DrawIcon	Method	Draws an icon.
DrawRectangle	Method	Draws a rectangle.
DrawStaticText	Method	Draws the static text.
DrawText	Method	Draws the text.
DrawTrendLine	Method	Draws a trend line.
DrawTriangle	Method	Draws a triangle.
DrawVerticalLine	Method	Draws a vertical line.
FindAllObjects	Method	Finds all the chart objects of the specified type.
FindObject	Method	Finds the chart object of the specified name.

Height	Property	Gets the hight of the chart area.
IsAlive	Property	Checks whether the instance is still on the chart.
MouseDown	Event	Occurs when the left mouse button is pressed down.
MouseEnter	Event	Occurs when the cursor hover over the chart area.
MouseLeave	Event	Occurs when the cursor leaves the chart area
MouseMove	Event	Occurs when the cursor moves over the chart area.
MouseUp	Event	Occurs when the left mouse button is released.
MouseWheel	Event	Occurs when the mouse wheel button is rotated.
ObjectAdded	Event	Occurs when a chart object is added to the chart area.
ObjectHoverChanged	Event	Occurs when the cursor hovers over or leaves the object.
ObjectRemoved	Event	Occurs when a chart object is removed from the chart area.
Objects	Property	Gets the chart objects collection.
ObjectSelectionChanged	Event	Occurs when a chart object is selected or deselected.
ObjectUpdated	Event	Occurs when a chart object is updated - one or several properties of the chart object have changed.
RemoveAllObjects	Method	Removes all interactive and non-interactive objects available for the cBot or Indicator.
RemoveObject	Method	Removes the chart object of the specified name.
ScrollChanged	Event	Occurs when the X-axis position value or the Y-axis position value changes while scrolling.
SetYRange	Method	Sets the Y-axis lowest and highest values range. Allows scrolling the chart by the Y-axis. If only one of the values is set, then the chart will be expanded regarding the lowest or highest value respectively.
SizeChanged	Event	Occurs when the chart area size has changed.
TopY	Property	Gets the highest visible Y-axis value.
Width	Property	Gets the width of the chart area.

IsAlive

Summary

Checks whether the instance is still on the chart.

Syntax

```
public bool IsAlive{ get; }
```

Width

Summary

Gets the width of the chart area.

Syntax

```
public double Width{ get; }
```

Height

Summary

Gets the hight of the chart area.

Syntax

```
public double Height{ get; }
```

BottomY

Summary

Gets the lowest visible Y-axis value.

Syntax

```
public double BottomY{ get; }
```

TopY

Summary

Gets the highest visible Y-axis value.

Syntax

```
public double TopY{ get; }
```

Objects

Summary

Gets the chart objects collection.

Syntax

```
public IReadonlyList Objects{ get; }
```

SetYRange

Summary

Sets the Y-axis lowest and highest values range. Allows scrolling the chart by the Y-axis. If only one of the values is set, then the chart will be expanded regarding the lowest or highest value respectively.

Syntax

```
public void SetYRange(double bottomY, double topY)
```

Parameters

Name	Description

FindAllObjects

Summary

Finds all the chart objects of the specified type.

Syntax

```
public T[] FindAllObjects()
```

```
public ChartObject[] FindAllObjects(ChartObjectType objectType)
```

FindAllObjects

Summary

Finds all the chart objects of the specified type.

Syntax

```
public T[] FindAllObjects()
```

```
public ChartObject[] FindAllObjects(ChartObjectType objectType)
```

Parameters

Name	Description

FindObject

Summary

Finds the chart object of the specified name.

Syntax

```
public ChartObject FindObject(string objectName)
```

Parameters

Name	Description

Example 1

```
// Draw a horizontal line.
Chart.DrawHorizontalLine("hLine", Symbol.Ask, Color.Red);
// Find the line that was drawn.
var obj = Chart.FindObject("hLine");
Print("Found object with name {0}", obj.Name);
```

RemoveObject

Summary

Removes the chart object of the specified name.

Syntax

```
public void RemoveObject(string objectName)
```

Parameters

Name	Description

RemoveAllObjects

Summary

Removes all interactive and non-interactive objects available for the cBot or Indicator.

Syntax

```
public void RemoveAllObjects()
```

DrawHorizontalLine

Summary

Draws a horizontal line.

Syntax

```
public ChartHorizontalLine DrawHorizontalLine(string name, double y, Color color)
```

```
public ChartHorizontalLine DrawHorizontalLine(string name, double y, Color color, int
thickness)
```

```
public ChartHorizontalLine DrawHorizontalLine(string name, double y, Color color, int
thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawHorizontalLine

Summary

Draws a horizontal line.

Syntax

```
public ChartHorizontalLine DrawHorizontalLine(string name, double y, Color color)
```

```
public ChartHorizontalLine DrawHorizontalLine(string name, double y, Color color, int
thickness)
```

```
public ChartHorizontalLine DrawHorizontalLine(string name, double y, Color color, int
thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawHorizontalLine

Summary

Draws a horizontal line.

Syntax

```
public ChartHorizontalLine DrawHorizontalLine(string name, double y, Color color)
```

```
public ChartHorizontalLine DrawHorizontalLine(string name, double y, Color color, int
thickness)
```

```
public ChartHorizontalLine DrawHorizontalLine(string name, double y, Color color, int
thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawVerticalLine

Summary

Draws a vertical line.

Syntax

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness, LineStyle lineStyle)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
```

```
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawVerticalLine

Summary

Draws a vertical line.

Syntax

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness, LineStyle lineStyle)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawVerticalLine

Summary

Draws a vertical line.

Syntax

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness, LineStyle lineStyle)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawVerticalLine

Summary

Draws a vertical line.

Syntax

```
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness, LineStyle lineStyle)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawVerticalLine

Summary

Draws a vertical line.

Syntax

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness, LineStyle lineStyle)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawVerticalLine

Summary

Draws a vertical line.

Syntax

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, DateTime time, Color color, int
thickness, LineStyle lineStyle)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
thickness)
```

```
public ChartVerticalLine DrawVerticalLine(string name, int barIndex, Color color, int
thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTrendLine

Summary

Draws a trend line.

Syntax

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTrendLine

Summary

Draws a trend line.

Syntax

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTrendLine

Summary

Draws a trend line.

Syntax

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTrendLine

Summary

Draws a trend line.

Syntax

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTrendLine

Summary

Draws a trend line.

Syntax

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTrendLine

Summary

Draws a trend line.

Syntax

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartTrendLine DrawTrendLine(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawEquidistantChannel

Summary

Draws an equidistant channel.

Syntax

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1, DateTime time2, double y2, double channelHeight, Color color)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1, DateTime time2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1, DateTime time2, double y2, double channelHeight, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1, int barIndex2, double y2, double channelHeight, Color color)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1, int barIndex2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1, int barIndex2, double y2, double channelHeight, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawEquidistantChannel

Summary

Draws an equidistant channel.

Syntax

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawEquidistantChannel

Summary

Draws an equidistant channel.

Syntax

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color)
```

```
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawEquidistantChannel

Summary

Draws an equidistant channel.

Syntax

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawEquidistantChannel

Summary

Draws an equidistant channel.

Syntax

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
```

```
int barIndex2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawEquidistantChannel

Summary

Draws an equidistant channel.

Syntax

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, DateTime time1, double y1,
DateTime time2, double y2, double channelHeight, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color, int thickness)
```

```
public ChartEquidistantChannel DrawEquidistantChannel(string name, int barIndex1, double y1,
int barIndex2, double y2, double channelHeight, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawText

Summary

Draws the text.

Syntax

```
public ChartText DrawText(string name, string text, DateTime time, double y, Color color)
```

```
public ChartText DrawText(string name, string text, int barIndex, double y, Color color)
```

Parameters

Name	Description

Example 1

```
// Draw the text on the last bar high.
var highPrice = MarketSeries.High.LastValue;
var openTime = MarketSeries.OpenTime.LastValue;
var text = Chart.DrawText("text1", "High is here", openTime, highPrice, Color.Red);
text.VerticalAlignment = VerticalAlignment.Bottom;
text.HorizontalAlignment = HorizontalAlignment.Center;
```

DrawText

Summary

Draws the text.

Syntax

```
public ChartText DrawText(string name, string text, DateTime time, double y, Color color)
```

```
public ChartText DrawText(string name, string text, int barIndex, double y, Color color)
```

Parameters

Name	Description

DrawStaticText

Summary

Draws the static text.

Syntax

```
public ChartStaticText DrawStaticText(string name, string text, VerticalAlignment
verticalAlignment, HorizontalAlignment horizontalAlignment, Color color)
```

Parameters

Name	Description

DrawIcon

Summary

Draws an icon.

Syntax

```
public ChartIcon DrawIcon(string name, ChartIconType iconType, DateTime time, double y, Color
color)
```

```
public ChartIcon DrawIcon(string name, ChartIconType iconType, int barIndex, double y, Color
color)
```

Parameters

Name	Description

DrawIcon

Summary

Draws an icon.

Syntax

```
public ChartIcon DrawIcon(string name, ChartIconType iconType, DateTime time, double y, Color color)
```

```
public ChartIcon DrawIcon(string name, ChartIconType iconType, int barIndex, double y, Color color)
```

Parameters

Name	Description

Example 1

```
// Draw an icon on the last bar high.
```

DrawFibonacciRetracement

Summary

Draws a Fibonacci retracement.

Syntax

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciRetracement

Summary

Draws a Fibonacci retracement.

Syntax

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciRetracement

Summary

Draws a Fibonacci retracement.

Syntax

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciRetracement

Summary

Draws a Fibonacci retracement.

Syntax

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciRetracement

Summary

Draws a Fibonacci retracement.

Syntax

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciRetracement

Summary

Draws a Fibonacci retracement.

Syntax

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciRetracement DrawFibonacciRetracement(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciExpansion

Summary

Draws a Fibonacci expansion.

Syntax

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawFibonacciExpansion

Summary

Draws a Fibonacci expansion.

Syntax

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawFibonacciExpansion

Summary

Draws a Fibonacci expansion.

Syntax

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawFibonacciExpansion

Summary

Draws a Fibonacci expansion.

Syntax

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciExpansion

Summary

Draws a Fibonacci expansion.

Syntax

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciExpansion

Summary

Draws a Fibonacci expansion.

Syntax

```
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartFibonacciExpansion DrawFibonacciExpansion(string name, int barIndex1, double y1,
int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawFibonacciFan

Summary

Draws a Fibonacci fan.

Syntax

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime
time2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime
time2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciFan

Summary

Draws a Fibonacci fan.

Syntax

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int
barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int
barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciFan

Summary

Draws a Fibonacci fan.

Syntax

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime
time2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime
time2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime
time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int
barIndex2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int
barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int
barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciFan

Summary

Draws a Fibonacci fan.

Syntax

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciFan

Summary

Draws a Fibonacci fan.

Syntax

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawFibonacciFan

Summary

Draws a Fibonacci fan.

Syntax

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartFibonacciFan DrawFibonacciFan(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawAndrewsPitchfork

Summary

Draws an Andrew's pitchfork.

Syntax

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawAndrewsPitchfork

Summary

Draws an Andrew's pitchfork.

Syntax

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
```

```
barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawAndrewsPitchfork

Summary

Draws an Andrew's pitchfork.

Syntax

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawAndrewsPitchfork

Summary

Draws an Andrew's pitchfork.

Syntax

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawAndrewsPitchfork

Summary

Draws an Andrews pitchfork.

Syntax

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawAndrewsPitchfork

Summary

Draws an Andrew's pitchfork.

Syntax

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, DateTime time1, double y1,
DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartAndrewsPitchfork DrawAndrewsPitchfork(string name, int barIndex1, double y1, int
barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle
lineStyle)
```

Parameters

Name	Description

DrawRectangle

Summary

Draws a rectangle.

Syntax

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawRectangle

Summary

Draws a rectangle.

Syntax

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawRectangle

Summary

Draws a rectangle.

Syntax

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawRectangle

Summary

Draws a rectangle.

Syntax

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawRectangle

Summary

Draws a rectangle.

Syntax

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawRectangle

Summary

Draws a rectangle.

Syntax

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, DateTime time1, double y1, DateTime time2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness)
```

```
public ChartRectangle DrawRectangle(string name, int barIndex1, double y1, int barIndex2,
double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawEllipse

Summary

Draws an ellipse.

Syntax

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double
y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double
y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double
y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawEllipse

Summary

Draws an ellipse.

Syntax

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double
y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawEllipse

Summary

Draws an ellipse.

Syntax

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double
y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double
y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double
y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double
y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double
y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double
y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawEllipse

Summary

Draws an ellipse.

Syntax

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawEllipse

Summary

Draws an ellipse.

Syntax

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawEllipse

Summary

Draws an ellipse.

Syntax

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, DateTime time1, double y1, DateTime time2, double y2, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness)
```

```
public ChartEllipse DrawEllipse(string name, int barIndex1, double y1, int barIndex2, double y2, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTriangle

Summary

Draws a triangle.

Syntax

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTriangle

Summary

Draws a triangle.

Syntax

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2, double y2, DateTime time3, double y3, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTriangle

Summary

Draws a triangle.

Syntax

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double
y2, int barIndex3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double
y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double
y2, int barIndex3, double y3, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTriangle

Summary

Draws a triangle.

Syntax

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double
y2, int barIndex3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double
y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double
y2, int barIndex3, double y3, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTriangle

Summary

Draws a triangle.

Syntax

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double
y2, int barIndex3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double
y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double
y2, int barIndex3, double y3, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

DrawTriangle

Summary

Draws a triangle.

Syntax

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, DateTime time1, double y1, DateTime time2,
double y2, DateTime time3, double y3, Color color, int thickness, LineStyle lineStyle)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness)
```

```
public ChartTriangle DrawTriangle(string name, int barIndex1, double y1, int barIndex2, double y2, int barIndex3, double y3, Color color, int thickness, LineStyle lineStyle)
```

Parameters

Name	Description

MouseEnter

Summary

Occurs when the cursor hover over the chart area.

Syntax

```
public event Action MouseEnter
```

MouseLeave

Summary

Occurs when the cursor leaves the chart area

Syntax

```
public event Action MouseLeave
```

MouseMove

Summary

Occurs when the cursor moves over the chart area.

Syntax

```
public event Action MouseMove
```

MouseDown

Summary

Occurs when the left mouse button is pressed down.

Syntax

```
public event Action MouseDown
```

MouseUp

Summary

Occurs when the left mouse button is released.

Syntax

```
public event Action MouseUp
```

MouseWheel

Summary

Occurs when the mouse wheel button is rotated.

Syntax

```
public event Action MouseWheel
```

DragStart

Summary

Occurs when MouseDown event is happening on a chart area and a mouse is captured for dragging.

Syntax

```
public event Action DragStart
```

DragEnd

Summary

Occurs when mouse button is released while dragging a chart area or a chart area loses mouse capture.

Syntax

```
public event Action DragEnd
```

Drag

Summary

Occurs when dragging a chart area.

Syntax

```
public event Action Drag
```

SizeChanged

Summary

Occurs when the chart area size has changed.

Syntax

```
public event Action SizeChanged
```

ScrollChanged

Summary

Occurs when the X-axis position value or the Y-axis position value changes while scrolling.

Syntax

```
public event Action ScrollChanged
```

ObjectAdded

Summary

Occurs when a chart object is added to the chart area.

Syntax

```
public event Action ObjectAdded
```

ObjectUpdated

Summary

Occurs when a chart object is updated - one or several properties of the chart object have changed.

Syntax

```
public event Action ObjectUpdated
```

ObjectRemoved

Summary

Occurs when a chart object is removed from the chart area.

Syntax

```
public event Action ObjectRemoved
```

ObjectSelectionChanged

Summary

Occurs when a chart object is selected or deselected.

Syntax

```
public event Action ObjectSelectionChanged
```

ObjectHoverChanged

Summary

Occurs when the cursor hovers over or leaves the object.

Syntax

```
public event Action ObjectHoverChanged
```

ChartColorEventArgs

Summary

Provides data for the chart color event.

Syntax

```
public class ChartColorEventArgs : Object
```

Members

Name	Type	Summary
Chart	Property	Gets the chart.

Chart

Summary

Gets the chart.

Syntax

```
public Chart Chart{ get; }
```

ChartColorSettings

Summary

Represents the charts Color Settings.

Remarks

Use the Color classes to set the chart Color Settings.

Syntax

```
public interface ChartColorSettings
```

Members

Name	Type	Summary
AskPriceLineColor	Property	Gets or sets the color of the ask price line.
BackgroundColor	Property	Gets or sets the color of the chart background.
BearFillColor	Property	Gets or sets the color of the bear candle fill.
BearOutlineColor	Property	Gets or sets the color of the bear candle or bar outline.
BidPriceLineColor	Property	Gets or sets the color of the bid price line.
BullFillColor	Property	Gets or sets the color of the bull candle fill.
BullOutlineColor	Property	Gets or sets the color of the bull candle or bar outline.
BuyColor	Property	Gets or sets the color of Buy positions and orders.
ForegroundColor	Property	Gets or sets the color of the chart foreground.
GridLinesColor	Property	Gets or sets the color of the grid lines.
LosingDealColor	Property	Gets or sets the color of the losing deal.
PeriodSeparatorColor	Property	Gets or sets the color of the period separator.
SellColor	Property	Gets or sets the color of Sell order positions and orders.
TickVolumeColor	Property	Gets or sets the color of the tick volume.
WinningDealColor	Property	Gets or sets the color of the winning deal.

BackgroundColor

Summary

Gets or sets the color of the chart background.

Syntax

```
public Color BackgroundColor{ get; set; }
```

ForegroundColor

Summary

Gets or sets the color of the chart foreground.

Syntax

```
public Color ForegroundColor{ get; set; }
```

GridLinesColor

Summary

Gets or sets the color of the grid lines.

Syntax

```
public Color GridLinesColor{ get; set; }
```

PeriodSeparatorColor

Summary

Gets or sets the color of the period separator.

Syntax

```
public Color PeriodSeparatorColor{ get; set; }
```

BullOutlineColor

Summary

Gets or sets the color of the bull candle or bar outline.

Syntax

```
public Color BullOutlineColor{ get; set; }
```

BearOutlineColor

Summary

Gets or sets the color of the bear candle or bar outline.

Syntax

```
public Color BearOutlineColor{ get; set; }
```

BullFillColor

Summary

Gets or sets the color of the bull candle fill.

Syntax

```
public Color BullFillColor{ get; set; }
```

BearFillColor

Summary

Gets or sets the color of the bear candle fill.

Syntax

```
public Color BearFillColor{ get; set; }
```

TickVolumeColor

Summary

Gets or sets the color of the tick volume.

Syntax

```
public Color TickVolumeColor{ get; set; }
```

WinningDealColor

Summary

Gets or sets the color of the winning deal.

Syntax

```
public Color WinningDealColor{ get; set; }
```

LosingDealColor

Summary

Gets or sets the color of the losing deal.

Syntax

```
public Color LosingDealColor{ get; set; }
```

AskPriceLineColor

Summary

Gets or sets the color of the ask price line.

Syntax

```
public Color AskPriceLineColor{ get; set; }
```

BidPriceLineColor

Summary

Gets or sets the color of the bid price line.

Syntax

```
public Color BidPriceLineColor{ get; set; }
```

BuyColor

Summary

Gets or sets the color of Buy positions and orders.

Syntax

```
public Color BuyColor{ get; set; }
```

SellColor

Summary

Gets or sets the color of Sell order positions and orders.

Syntax

```
public Color SellColor{ get; set; }
```

ChartDisplaySettings

Summary

Represents the chart display settings.

Syntax

```
public interface ChartDisplaySettings
```

Members

Name	Type	Summary
AskPriceLine	Property	Gets or sets the ask price line.
BidPriceLine	Property	Gets or sets the bid price line.
ChartScale	Property	Gets or sets the chart scale.
DealMap	Property	Gets or sets a value indicating the deal map.
Grid	Property	Gets or sets the grid.
MarketSentiment	Property	Gets or sets the market sentiment index.
Orders	Property	Gets or sets the orders.
PeriodSeparators	Property	Gets or sets the period separators.
Positions	Property	Gets or sets the positions.
PriceAlerts	Property	Gets or sets the price alerts.
PriceAxisOverlayButtons	Property	Gets or sets the price axis overlay buttons.
Targets	Property	Gets or sets the targets.
TickVolume	Property	Gets or sets the tick volume.

Positions

Summary

Gets or sets the positions.

Syntax

```
public bool Positions{ get; set; }
```

Orders

Summary

Gets or sets the orders.

Syntax

```
public bool Orders{ get; set; }
```

BidPriceLine

Summary

Gets or sets the bid price line.

Syntax

```
public bool BidPriceLine{ get; set; }
```

AskPriceLine

Summary

Gets or sets the ask price line.

Syntax

```
public bool AskPriceLine{ get; set; }
```

Grid

Summary

Gets or sets the grid.

Syntax

```

```

```
public bool Grid{ get; set; }
```

PeriodSeparators

Summary

Gets or sets the period separators.

Syntax

```
public bool PeriodSeparators{ get; set; }
```

TickVolume

Summary

Gets or sets the tick volume.

Syntax

```
public bool TickVolume{ get; set; }
```

DealMap

Summary

Gets or sets a value indicating the deal map.

Syntax

```
public bool DealMap{ get; set; }
```

ChartScale

Summary

Gets or sets the chart scale.

Syntax

```
public bool ChartScale{ get; set; }
```

PriceAxisOverlayButtons

Summary

Gets or sets the price axis overlay buttons.

Syntax

```
public bool PriceAxisOverlayButtons{ get; set; }
```

PriceAlerts

Summary

Gets or sets the price alerts.

Syntax

```
public bool PriceAlerts{ get; set; }
```

MarketSentiment

Summary

Gets or sets the market sentiment index.

Syntax

```
public bool MarketSentiment{ get; set; }
```

Targets

Summary

Gets or sets the targets.

Syntax

```
public bool Targets{ get; set; }
```

ChartDisplaySettingsEventArgs

Summary

Provides data for the chart display settings event.

Syntax

```
public class ChartDisplaySettingsEventArgs : Object
```

Members

Name	Type	Summary
Chart (2)	Property	Gets the chart.

Chart

Summary

Gets the chart.

Syntax

```

```

```
public Chart Chart{ get; }
```

ChartDragEventArgs

Summary

Provides data for the chart dragging event.

Syntax

```
public class ChartDragEventArgs : ChartMouseEventArgs
```

Members

Name	Type	Summary

ChartEllipse

Summary

Represent the Ellipse chart object.

Syntax

```
public interface ChartEllipse : ChartShape, ChartObject
```

Members

Name	Type	Summary
Time1 (2)	Property	Gets or sets the value 1 on the Time line.
Time2 (2)	Property	Gets or sets the value 2 on the Time line.
Y1 (2)	Property	Gets or sets the value 1 on the Y-axis.
Y2 (2)	Property	Gets or sets the value 2 on the Y-axis.

Time1

Summary

Gets or sets the value 1 on the Time line.

Syntax

```
public DateTime Time1{ get; set; }
```

Time2

Summary

Gets or sets the value 2 on the Time line.

Syntax

```
public DateTime Time2{ get; set; }
```

Y1

Summary

Gets or sets the value 1 on the Y-axis.

Syntax

```
public double Y1{ get; set; }
```

Y2

Summary

Gets or sets the value 2 on the Y-axis.

Syntax

```
```

```
public double Y2{ get; set; }
```

ChartEquidistantChannel

Summary

Represents the Equidistant Channel chart object. The tool that allows drawing two precisely parallel lines in any direction on the chart.

Syntax

```
public interface ChartEquidistantChannel : ChartObject
```

Members

Name	Type	Summary
ChannelHeight	Property	Gets or sets the height of the Equidistant Channel.
Color (2)	Property	Gets or sets the Equidistant Channel line color.
ExtendToInfinity	Property	Defines if the Equidistant channel extends to infinity.
LineStyle (2)	Property	Gets or sets the Equidistant channel line style.
ShowAngle	Property	Gets or sets the Equidistant Channel angle.
Thickness (2)	Property	Gets or sets the Equidistant Channel line thickness.
Time1 (3)	Property	Gets or sets the value 1 on the Time line.
Time2 (3)	Property	Gets or sets the value 2 on the Time line.
Y1 (3)	Property	Gets or sets the value 1 on the Y-axis.
Y2 (3)	Property	Gets or sets the value 2 on the Y-axis.

Time1

Summary

Gets or sets the value 1 on the Time line.

Syntax

```
public DateTime Time1{ get; set; }
```

Time2

Summary

Gets or sets the value 2 on the Time line.

Syntax

```
public DateTime Time2{ get; set; }
```

Y1

Summary

Gets or sets the value 1 on the Y-axis.

Syntax

```
public double Y1{ get; set; }
```

Y2

Summary

Gets or sets the value 2 on the Y-axis.

Syntax

```
public double Y2{ get; set; }
```

ChannelHeight

Summary

Gets or sets the height of the Equidistant Channel.

Syntax

```
public double ChannelHeight{ get; set; }
```

Thickness

Summary

Gets or sets the Equidistant Channel line thickness.

Syntax

```
public int Thickness{ get; set; }
```

LineStyle

Summary

Gets or sets the Equidistant channel line style.

Syntax

```
public LineStyle LineStyle{ get; set; }
```

Color

Summary

Gets or sets the Equidistant Channel line color.

Syntax

```

```

```
public Color Color{ get; set; }
```

ShowAngle

Summary

Gets or sets the Equidistant Channel angle.

Syntax

```
public bool ShowAngle{ get; set; }
```

ExtendToInfinity

Summary

Defines if the Equidistant channel extends to infinity.

Syntax

```
public bool ExtendToInfinity{ get; set; }
```

ChartFibonacciBase

Summary

Represents the Fibonacci tools options.

Syntax

```
public interface ChartFibonacciBase : ChartObject
```

Members

Name	Type	Summary

Color (3)	Property	Gets or sets the lines color.
DisplayPrices	Property	Defines if the Fibonacci levels display the prices
FibonacciLevels	Property	Gets the Fibonacci levels.
LineStyle (3)	Property	Gets or sets the lines style.
Thickness (3)	Property	Gets or sets the lines thickness.

FibonacciLevels

Summary

Gets the Fibonacci levels.

Syntax

```
public IReadonlyList FibonacciLevels{ get; }
```

DisplayPrices

Summary

Defines if the Fibonacci levels display the prices

Syntax

```
public bool DisplayPrices{ get; set; }
```

Thickness

Summary

Gets or sets the lines thickness.

Syntax

```
public int Thickness{ get; set; }
```

Color

Summary

Gets or sets the lines color.

Syntax

```
public Color Color{ get; set; }
```

LineStyle

Summary

Gets or sets the lines style.

Syntax

```
public LineStyle LineStyle{ get; set; }
```

ChartFibonacciExpansion

Summary

Represents the Fibonacci Expansion chart object.

Syntax

```
public interface ChartFibonacciExpansion : ChartFibonacciBase, ChartObject
```

Members

Name	Type	Summary
Time1 (4)	Property	Gets or sets the value 1 on the Time line.
Time2 (4)	Property	Gets or sets the value 2 on the Time line.

Time3 (2)	Property	Gets or sets the value 3 on the Time line.
Y1 (4)	Property	Gets or sets the value 1 on the Y-axis.
Y2 (4)	Property	Gets or sets the value 2 on the Y-axis.
Y3 (2)	Property	Gets or sets the value 3 on the Y-axis.

Time1

Summary

Gets or sets the value 1 on the Time line.

Syntax

```
public DateTime Time1{ get; set; }
```

Time2

Summary

Gets or sets the value 2 on the Time line.

Syntax

```
public DateTime Time2{ get; set; }
```

Time3

Summary

Gets or sets the value 3 on the Time line.

Syntax

```
public DateTime Time3{ get; set; }
```

Y1

Summary

Gets or sets the value 1 on the Y-axis.

Syntax

```
public double Y1{ get; set; }
```

Y2

Summary

Gets or sets the value 2 on the Y-axis.

Syntax

```
public double Y2{ get; set; }
```

Y3

Summary

Gets or sets the value 3 on the Y-axis.

Syntax

```
public double Y3{ get; set; }
```

ChartFibonacciFan

Summary

Represents the Fibonacci Fan chart object.

Syntax

```
public interface ChartFibonacciFan : ChartFibonacciBase, ChartObject
```

Members

Name	Type	Summary
Time1 (5)	Property	Gets or sets the value 1 on the Time line.
Time2 (5)	Property	Gets or sets the value 2 on the Time line.
Y1 (5)	Property	Gets or sets the value 1 on the Y-axis.
Y2 (5)	Property	Gets or sets the value 2 on the Y-axis.

Time1

Summary

Gets or sets the value 1 on the Time line.

Syntax

```
public DateTime Time1{ get; set; }
```

Time2

Summary

Gets or sets the value 2 on the Time line.

Syntax

```
public DateTime Time2{ get; set; }
```

Y1

Summary

Gets or sets the value 1 on the Y-axis.

Syntax

```
public double Y1{ get; set; }
```

Y2

Summary

Gets or sets the value 2 on the Y-axis.

Syntax

```
public double Y2{ get; set; }
```

ChartFibonacciRetracement

Summary

Represents the Fibonacci Retracement chart object.

Syntax

```
public interface ChartFibonacciRetracement : ChartFibonacciBase, ChartObject
```

Members

Name	Type	Summary
Time1 (6)	Property	Gets or sets the value 1 on the Time line.
Time2 (6)	Property	Gets or sets the value 2 on the Time line.
Y1 (6)	Property	Gets or sets the value 1 on the Y-axis.
Y2 (6)	Property	Gets or sets the value 2 on the Y-axis.

Time1

Summary

Gets or sets the value 1 on the Time line.

Syntax

```
public DateTime Time1{ get; set; }
```

Time2

Summary

Gets or sets the value 2 on the Time line.

Syntax

```
public DateTime Time2{ get; set; }
```

Y1

Summary

Gets or sets the value 1 on the Y-axis.

Syntax

```
public double Y1{ get; set; }
```

Y2

Summary

Gets or sets the value 2 on the Y-axis.

Syntax

```
public double Y2{ get; set; }
```

ChartHorizontalLine

Summary

Represents the Horizontal Line chart object. Used to mark a certain value on the Y-axis throughout the whole chart.

Syntax

```
public interface ChartHorizontalLine : ChartObject
```

Members

Name	Type	Summary
Color (4)	Property	Gets or sets the line color.
LineStyle (4)	Property	Gets or sets the line style.
Thickness (4)	Property	Gets or sets the line thickness.
Y	Property	Gets or sets the Y-axis value of the line location.

Y

Summary

Gets or sets the Y-axis value of the line location.

Syntax

```
public double Y{ get; set; }
```

Thickness

Summary

Gets or sets the line thickness.

Syntax

```
public int Thickness{ get; set; }
```

Color

Summary

Gets or sets the line color.

Syntax

```
public Color Color{ get; set; }
```

LineStyle

Summary

Gets or sets the line style.

Syntax

```
public LineStyle LineStyle{ get; set; }
```

ChartIcon

Summary

Represents the Icon chart object.

Syntax

```
```

```
public interface ChartIcon : ChartObject
```

Members

Name	Type	Summary
Color (5)	Property	Gets or sets the color of the icon.
IconType	Property	Gets or sets the type of the icon.
Time	Property	Gets or sets the Time value of the icon location.
Y (2)	Property	Gets or sets the Y-axis value of the icon location.

IconType

Summary

Gets or sets the type of the icon.

Syntax

```
public ChartIconType IconType{ get; set; }
```

Time

Summary

Gets or sets the Time value of the icon location.

Syntax

```
public DateTime Time{ get; set; }
```

Y

Summary

Gets or sets the Y-axis value of the icon location.

Syntax

```
public double Y{ get; set; }
```

Color

Summary

Gets or sets the color of the icon.

Syntax

```
public Color Color{ get; set; }
```

ChartIconType

Summary

Represents the type of the Icon.

Syntax

```
public sealed enum ChartIconType
```

Members

Name	Type	Summary
Circle	Field	The Circle.
Diamond	Field	The Diamond.
DownArrow	Field	The Down Arrow.
DownTriangle	Field	The Down Triangle.
Square	Field	The Square.
Star	Field	The Star.
UpArrow	Field	The Up Arrow.
UpTriangle	Field	The Up Triangle.

UpArrow

Summary

The Up Arrow.

Syntax

```
ChartIconType.UpArrow
```

DownArrow

Summary

The Down Arrow.

Syntax

```
ChartIconType.DownArrow
```

Circle

Summary

The Circle.

Syntax

```
ChartIconType.Circle
```

Square

Summary

The Square.

Syntax

```
ChartIconType.Square
```

Diamond

Summary

The Diamond.

Syntax

```
ChartIconType.Diamond
```

Star

Summary

The Star.

Syntax

```
ChartIconType.Star
```

UpTriangle

Summary

The Up Triangle.

Syntax

```
ChartIconType.UpTriangle
```

DownTriangle

Summary

The Down Triangle.

Syntax

```
ChartIconType.DownTriangle
```

ChartMouseEventArgs

Summary

Provides data for the mouse related routed events.

Syntax

```
public class ChartMouseEventArgs : Object
```

Members

Name	Type	Summary
AltKey	Property	Defines whether the Alt key is pressed during the mouse event.
BarIndex	Property	Gets the exact bar index of the mouse event.
Chart (3)	Property	Gets the chart.
ChartArea	Property	Gets the chart area.
CtrlKey	Property	
MouseX	Property	Gets the X-axis value of the mouse event.
MouseY	Property	Gets the Y-axis value of the mouse event.
ShiftKey	Property	Defines whether the Shift key is pressed during the mouse event.
TimeValue	Property	Gets the time value on the X-axis where the mouse event occurs.
YValue	Property	Gets the Y-axis value of the mouse event.

Chart

Summary

Gets the chart.

Syntax

```
public Chart Chart{ get; }
```

ChartArea

Summary

Gets the chart area.

Syntax

```
public ChartArea ChartArea{ get; }
```

MouseX

Summary

Gets the X-axis value of the mouse event.

Syntax

```
public double MouseX{ get; }
```

MouseY

Summary

Gets the Y-axis value of the mouse event.

Syntax

```
public double MouseY{ get; }
```

TimeValue

Summary

Gets the time value on the X-axis where the mouse event occurs.

Syntax

```
public DateTime TimeValue{ get; }
```

BarIndex

Summary

Gets the exact bar index of the mouse event.

Syntax

```
public double BarIndex{ get; }
```

YValue

Summary

Gets the Y-axis value of the mouse event.

Syntax

```
public double YValue{ get; }
```

CtrlKey

Syntax

```
public bool CtrlKey{ get; }
```

ShiftKey

Summary

Defines whether the Shift key is pressed during the mouse event.

Syntax

```
public bool ShiftKey{ get; }
```

AltKey

Summary

Defines whether the Alt key is pressed during the mouse event.

Syntax

```
public bool AltKey{ get; }
```

ChartMouseWheelEventArgs

Summary

Provides data for the mouse wheel scroll event.

Syntax

```
public class ChartMouseWheelEventArgs : ChartMouseEventArgs
```

Members

Name	Type	Summary
Delta	Property	Gets the number of detents the mouse wheel has rotated. A detent is one notch of the mouse wheel.

Delta

Summary

Gets the number of detents the mouse wheel has rotated. A detent is one notch of the mouse wheel.

Syntax

```
public int Delta{ get; }
```

ChartObject

Summary

Represents the chart object.

Syntax

```
public interface ChartObject
```

Members

Name	Type	Summary
Comment	Property	Gets or sets the comment for the chart object.
IsAlive (2)	Property	Defines if the chart object still exists on the chart.
IsInteractive	Property	Defines whether the instance is interactive. The non-interactive chart objects cannot be selected, have no hover effect and cannot be searched. Available only to the current cBot or Indicator and will be removed when the cBot/Indicator stops TBD
Name	Property	Gets the chart object name - the unique identifier for the object in the current chart area.
ObjectType	Property	Gets the chart object type.

ZIndex	Property	Gets or sets the location of a chart object on the Z-axis in respect to the other chart objects.

Name

Summary

Gets the chart object name - the unique identifier for the object in the current chart area.

Syntax

```
public string Name{ get; }
```

Comment

Summary

Gets or sets the comment for the chart object.

Syntax

```
public string Comment{ get; set; }
```

ObjectType

Summary

Gets the chart object type.

Syntax

```
public ChartObjectType ObjectType{ get; }
```

IsInteractive

Summary

Defines whether the instance is interactive. The non-interactive chart objects cannot be selected, have no hover effect and cannot be searched. Available only to the current cBot or Indicator and will be removed when the cBot/Indicator stops TBD

Syntax

```
public bool IsInteractive{ get; set; }
```

ZIndex

Summary

Gets or sets the location of a chart object on the Z-axis in respect to the other chart objects.

Syntax

```
public int ZIndex{ get; set; }
```

IsAlive

Summary

Defines if the chart object still exists on the chart.

Syntax

```
public bool IsAlive{ get; }
```

ChartObjectAddedEventArgs

Summary

Provides data for the adding chart object event.

Syntax

```
public class ChartObjectAddedEventArgs : ChartObjectEventArgs
```

Members

Name	Type	Summary

ChartObjectEventArgs

Syntax

```
public class ChartObjectEventArgs : Object
```

Members

Name	Type	Summary
Area	Property	Gets the chart area.
Chart (4)	Property	Gets the chart.
ChartObject	Property	Gets the chart object.

Chart

Summary

Gets the chart.

Syntax

```
public Chart Chart{ get; }
```

Area

Summary

Gets the chart area.

Syntax

```
public ChartArea Area{ get; }
```

ChartObject

Summary

Gets the chart object.

Syntax

```
public ChartObject ChartObject{ get; }
```

ChartObjectHoverChangedEventArgs

Summary

Provides data for the mouse hover over a chart object event.

Syntax

```
public class ChartObjectHoverChangedEventArgs : ChartObjectEventArgs
```

Members

Name	Type	Summary
IsObjectHovered	Property	Defines if the mouse is hovered over the chart object.

IsObjectHovered

Summary

Defines if the mouse is hovered over the chart object.

Syntax

```
public bool IsObjectHovered{ get; }
```

ChartObjectRemovedEventArgs

Summary

Provides data for the removing chart object event.

Syntax

```
public class ChartObjectRemovedEventArgs : ChartObjectEventArgs
```

Members

Name	Type	Summary

ChartObjectSelectionChangedEventArgs

Summary

Provides data for the chart object selecting or deselecting event.

Syntax

```
public class ChartObjectSelectionChangedEventArgs : ChartObjectEventArgs
```

Members

Name	Type	Summary
IsObjectSelected	Property	Defines whether the chart object is selected or deselected.

IsObjectSelected

Summary

Defines whether the chart object is selected or deselected.

Syntax

```
public bool IsObjectSelected{ get; }
```

ChartObjectType

Summary

The chart object types.

Syntax

```
public sealed enum ChartObjectType
```

Members

Name	Type	Summary
AndrewsPitchfork	Field	The Andrews Pitchfork that can be placed directly in the chart, bound to X-Y axises.
Ellipse	Field	The ellipse of any preferable size and rotation that can be drawn directly in the chart, bound to X-Y axises.
EquidistantChannel	Field	The equidistant channel that can be placed directly in the chart, bound to X-Y axises.
FibonacciExpansion	Field	The Fibonacci Expansion that can be placed directly in the chart, bound to X-Y axises. - a charting technique used to plot possible levels of support and resistance by tracking not only the primary trend but also the retracement.
FibonacciFan	Field	The Fibonacci Fan that can be placed directly in the chart, bound to X-Y axises. a charting technique used to estimate support and resistance levels by drawing the new trend lines based on the Fibonacci Retracement levels.
FibonacciRetracement	Field	The Fibonacci Retracement that can be placed directly in the chart, bound to X-Y axises. - a charting technique that uses the Fibonacci ratios to indicate the areas of support or resistance.
HorizontalLine	Field	The horizontal line. The line parallel to the X-axis that can be set on any Y-axis value.
Icon	Field	The icon. The collection of icons that can be placed directly in the chart, bound to X-Y axises.
Rectangle	Field	The rectangle of any preferable size and rotation that can be drawn directly in the chart, bound to X-Y axises.
StaticText	Field	
Text	Field	The text that can be placed directly in the chart, bound to X-Y axises.
TrendLine	Field	The trend line. The line with the start and end points that can be drawn in any direction on the chart.

Triangle	Field	The triangle of any preferable size and rotation that can be drawn directly in the chart, bound to X-Y axises.
VerticalLine	Field	The vertical line. The line parallel to the Y-axis that can be set on any X-axis value. used to mark certain time event or chart bar on the chart.TBD

HorizontalLine

Summary

The horizontal line. The line parallel to the X-axis that can be set on any Y-axis value.

Syntax

```
ChartObjectType.HorizontalLine
```

VerticalLine

Summary

The vertical line. The line parallel to the Y-axis that can be set on any X-axis value. used to mark certain time event or chart bar on the chart.TBD

Syntax

```
ChartObjectType.VerticalLine
```

TrendLine

Summary

The trend line. The line with the start and end points that can be drawn in any direction on the chart.

Syntax

```
ChartObjectType.TrendLine
```

Text

Summary

The text that can be placed directly in the chart, bound to X-Y axises.

Syntax

```
ChartObjectType.Text
```

StaticText

Syntax

```
ChartObjectType.StaticText
```

Icon

Summary

The icon. The collection of icons that can be placed directly in the chart, bound to X-Y axises.

Syntax

```
ChartObjectType.Icon
```

FibonacciRetracement

Summary

The Fibonacci Retracement that can be placed directly in the chart, bound to X-Y axises. - a charting technique that uses the Fibonacci ratios to indicate the areas of support or resistance.

Syntax

```
ChartObjectType.FibonacciRetracement
```

FibonacciExpansion

Summary

The Fibonacci Expansion that can be placed directly in the chart, bound to X-Y axises. - a charting technique used to plot possible levels of support and resistance by tracking not only the primary trend but also the retracement.

Syntax

```
ChartObjectType.FibonacciExpansion
```

FibonacciFan

Summary

The Fibonacci Fan that can be placed directly in the chart, bound to X-Y axises. a charting technique used to estimate support and resistance levels by drawing the new trend lines based on the Fibonacci Retracement levels.

Syntax

```
ChartObjectType.FibonacciFan
```

AndrewsPitchfork

Summary

The Andrews Pitchfork that can be placed directly in the chart, bound to X-Y axises.

Syntax

```
ChartObjectType.AndrewsPitchfork
```

Rectangle

Summary

The rectangle of any preferable size and rotation that can be drawn directly in the chart, bound to X-Y axises.

Syntax

```
ChartObjectType.Rectangle
```

Ellipse

Summary

The ellipse of any preferable size and rotation that can be drawn directly in the chart, bound to X-Y axises.

Syntax

```
ChartObjectType.Ellipse
```

Triangle

Summary

The triangle of any preferable size and rotation that can be drawn directly in the chart, bound to X-Y axises.

Syntax

```
ChartObjectType.Triangle
```

EquidistantChannel

Summary

The equidistant channel that can be placed directly in the chart, bound to X-Y axises.

Syntax

```
ChartObjectType.EquidistantChannel
```

ChartObjectUpdatedEventArgs

Summary

Provides data for the chart object update event.

Syntax

```
public class ChartObjectUpdatedEventArgs : ChartObjectEventArgs
```

Members

Name	Type	Summary

ChartRectangle

Summary

Represents the Rectangle chart object. A rectangle of any preferable size and rotation that can be drawn directly in the chart, bound to X-Y axises.

Syntax

```
public interface ChartRectangle : ChartShape, ChartObject
```

Members

Name	Type	Summary
Time1 (7)	Property	Gets or sets the value 1 on the Time line.
Time2 (7)	Property	Gets or sets the value 2 on the Time line.
Y1 (7)	Property	Gets or sets the value 1 on the Y-axis.
Y2 (7)	Property	Gets or sets the value 2 on the Y-axis.

Time1

Summary

Gets or sets the value 1 on the Time line.

Syntax

```
public DateTime Time1{ get; set; }
```

Time2

Summary

Gets or sets the value 2 on the Time line.

Syntax

```
public DateTime Time2{ get; set; }
```

Y1

Summary

Gets or sets the value 1 on the Y-axis.

Syntax

```
public double Y1{ get; set; }
```

Y2

Summary

Gets or sets the value 2 on the Y-axis.

Syntax

```
public double Y2{ get; set; }
```

ChartScrollEventArgs

Summary

Provides data for the chart scrolling event.

Syntax

```
public class ChartScrollEventArgs : Object
```

Members

Name	Type	Summary
BarsDelta	Property	Gets a value that indicates the amount of bars that the chart was scrolled for
BottomYDelta	Property	Gets the lowest value of the bars delta.
Chart (5)	Property	Gets the chart.
ChartArea (2)	Property	Gets the chart area.
TopYDelta	Property	Gets the highest value of the bars delta.

Chart

Summary

Gets the chart.

Syntax

```
public Chart Chart{ get; }
```

ChartArea

Summary

Gets the chart area.

Syntax

```
public ChartArea ChartArea{ get; }
```

BarsDelta

Summary

Gets a value that indicates the amount of bars that the chart was scrolled for

Syntax

```
public int BarsDelta{ get; }
```

BottomYDelta

Summary

Gets the lowest value of the bars delta.

Syntax

```
public double BottomYDelta{ get; }
```

TopYDelta

Summary

Gets the highest value of the bars delta.

Syntax

```
public double TopYDelta{ get; }
```

ChartShape

Summary

Represents the Shape chart object. Allows drawing a Rectangle, a Triangle, and an Ellipse on the chart.

Syntax

```
public interface ChartShape : ChartObject
```

Members

Name	Type	Summary
Color (6)	Property	Gets or sets the line color.
IsFilled	Property	Defines if the shape is filled.
LineStyle (5)	Property	Gets or sets the line style.
Thickness (5)	Property	Gets or sets the line thickness.

Thickness

Summary

Gets or sets the line thickness.

Syntax

```
public int Thickness{ get; set; }
```

LineStyle

Summary

Gets or sets the line style.

Syntax

```
public LineStyle LineStyle{ get; set; }
```

Color

Summary

Gets or sets the line color.

Syntax

```
public Color Color{ get; set; }
```

IsFilled

Summary

Defines if the shape is filled.

Syntax

```
public bool IsFilled{ get; set; }
```

ChartSizeEventArgs

Summary

Provides data for the chart size change event.

Syntax

```
public class ChartSizeEventArgs : Object
```

Members

Name	Type	Summary
Area (2)	Property	Gets the chart area.
Chart (6)	Property	Gets the chart.

Chart

Summary

Gets the chart.

Syntax

```
public Chart Chart{ get; }
```

Area

Summary

Gets the chart area.

Syntax

```
public ChartArea Area{ get; }
```

ChartStaticText

Syntax

```
public interface ChartStaticText : ChartObject
```

Members

Name	Type	Summary

Color (7)	Property	
HorizontalAlignment	Property	
Text (2)	Property	
VerticalAlignment	Property	

Color

Syntax

```
public Color Color{ get; set; }
```

Text

Syntax

```
public string Text{ get; set; }
```

VerticalAlignment

Syntax

```
public VerticalAlignment VerticalAlignment{ get; set; }
```

HorizontalAlignment

Syntax

```
public HorizontalAlignment HorizontalAlignment{ get; set; }
```

ChartText

Summary

Represents the Text chart object. Allows place the text anywhere on the chart, bound to the chart.

Syntax

```
public interface ChartText : ChartObject
```

Members

Name	Type	Summary
Color (8)	Property	Gets or sets the text color.
HorizontalAlignment (2)	Property	Gets or sets the horizontal alignment of the text regarding the anchor point.
Text (3)	Property	Gets or sets the text content.
Time (2)	Property	Gets or sets the Time line value.
VerticalAlignment (2)	Property	Gets or sets the vertical alignment of the text regarding the anchor point.
Y (3)	Property	Gets or sets the Y-axis value.

Time

Summary

Gets or sets the Time line value.

Syntax

```
public DateTime Time{ get; set; }
```

Y

Summary

Gets or sets the Y-axis value.

Syntax

```
public double Y{ get; set; }
```

Color

Summary

Gets or sets the text color.

Syntax

```
public Color Color{ get; set; }
```

Text

Summary

Gets or sets the text content.

Syntax

```
public string Text{ get; set; }
```

VerticalAlignment

Summary

Gets or sets the vertical alignment of the text regarding the anchor point.

Syntax

```
public VerticalAlignment VerticalAlignment{ get; set; }
```

HorizontalAlignment

Summary

Gets or sets the horizontal alignment of the text regarding the anchor point.

Syntax

```
public HorizontalAlignment HorizontalAlignment{ get; set; }
```

ChartTrendLine

Summary

Represents the Trend Line chart object. A straight line that can be drawn from point 1 to the point 2 in any direction to mark the trends on the chart.

Syntax

```
public interface ChartTrendLine : ChartObject
```

Members

Name	Type	Summary
CalculateY	Method	Calculates Y-axis value corresponding the specified bar index.
Color (9)	Property	Gets or sets the color of the Trend Line.
ExtendToInfinity (2)	Property	Defines if the Trend Line extends to infinity.
LineStyle (6)	Property	Gets or sets the Trend Line style.
ShowAngle (2)	Property	Defines the trend line angle.
Thickness (6)	Property	Gets or sets the thickness of the Trend Line.
Time1 (8)	Property	Gets or sets the value 1 on the Time line.
Time2 (8)	Property	Gets or sets the value 2 on the Time line.
Y1 (8)	Property	Gets or sets the value 1 on the Y-axis.
Y2 (8)	Property	Gets or sets the value 2 on the Y-axis.

Time1

Summary

Gets or sets the value 1 on the Time line.

Syntax

```
public DateTime Time1{ get; set; }
```

Time2

Summary

Gets or sets the value 2 on the Time line.

Syntax

```
public DateTime Time2{ get; set; }
```

Y1

Summary

Gets or sets the value 1 on the Y-axis.

Syntax

```
public double Y1{ get; set; }
```

Y2

Summary

Gets or sets the value 2 on the Y-axis.

Syntax

```
public double Y2{ get; set; }
```

Color

Summary

Gets or sets the color of the Trend Line.

Syntax

```
public Color Color{ get; set; }
```

Thickness

Summary

Gets or sets the thickness of the Trend Line.

Syntax

```
public int Thickness{ get; set; }
```

LineStyle

Summary

Gets or sets the Trend Line style.

Syntax

```
public LineStyle LineStyle{ get; set; }
```

ShowAngle

Summary

Defines the trend line angle.

Syntax

```
public bool ShowAngle{ get; set; }
```

ExtendToInfinity

Summary

Defines if the Trend Line extends to infinity.

Syntax

```
public bool ExtendToInfinity{ get; set; }
```

CalculateY

Summary

Calculates Y-axis value corresponding the specified bar index.

Syntax

```
public double CalculateY(int barIndex)
```

```
public double CalculateY(DateTime time)
```

Parameters

Name	Description

CalculateY

Summary

Calculates Y-axis value corresponding the specified time value.

Syntax

```
public double CalculateY(int barIndex)
```

```
public double CalculateY(DateTime time)
```

Parameters

Name	Description

ChartTriangle

Summary

Represents the Triangle chart object.

Syntax

```
public interface ChartTriangle : ChartShape, ChartObject
```

Members

Name	Type	Summary
Time1 (9)	Property	Gets or sets the value 1 on the Time line.
Time2 (9)	Property	Gets or sets the value 2 on the Time line.
Time3 (3)	Property	Gets or sets the value 3 on the Time line.
Y1 (9)	Property	Gets or sets the value 1 on the Y-axis.
Y2 (9)	Property	Gets or sets the value 2 on the Y-axis.
Y3 (3)	Property	Gets or sets the value 3 on the Y-axis.

Time1

Summary

Gets or sets the value 1 on the Time line.

Syntax

```
public DateTime Time1{ get; set; }
```

Time2

Summary

Gets or sets the value 2 on the Time line.

Syntax

```
public DateTime Time2{ get; set; }
```

Time3

Summary

Gets or sets the value 3 on the Time line.

Syntax

```
public DateTime Time3{ get; set; }
```

Y1

Summary

Gets or sets the value 1 on the Y-axis.

Syntax

```
public double Y1{ get; set; }
```

Y2

Summary

Gets or sets the value 2 on the Y-axis.

Syntax

```
public double Y2{ get; set; }
```

Y3

Summary

Gets or sets the value 3 on the Y-axis.

Syntax

```
public double Y3{ get; set; }
```

ChartType

Summary

Represents the predefined chart types.

Syntax

```
public sealed enum ChartType
```

Members

Name	Type	Summary
Bars	Field	The Bar chart.
Candlesticks	Field	The Candlestick chart.
Dots	Field	The Dots chart.
Line	Field	The Line chart.

Bars

Summary

The Bar chart.

Syntax

```
ChartType.Bars
```

Candlesticks

Summary

The Candlestick chart.

Syntax

```
ChartType.Candlesticks
```

Line

Summary

The Line chart.

Syntax

```
```

```
ChartType.Line
```

Dots

Summary

The Dots chart.

Syntax

```
ChartType.Dots
```

ChartTypeEventArgs

Summary

Provides data for the chart type chage event.

Syntax

```
public class ChartTypeEventArgs : Object
```

Members

Name	Type	Summary
Chart (7)	Property	Gets the chart.

Chart

Summary

Gets the chart.

Syntax

```

```

```
public Chart Chart{ get; }
```

ChartVerticalLine

Summary

Represents the Vertical Line chart object. The line parallel to the Y-axis that can be set on a certain time value on the X-axis.

Syntax

```
public interface ChartVerticalLine : ChartObject
```

Members

Name	Type	Summary
Color (10)	Property	Gets or sets the line color.
LineStyle (7)	Property	Gets or sets the line style.
Thickness (7)	Property	Gets or sets the line thickness.
Time (3)	Property	Gets or sets the value on the Time line.

Time

Summary

Gets or sets the value on the Time line.

Syntax

```
public DateTime Time{ get; set; }
```

Color

Summary

Gets or sets the line color.

Syntax

```
public Color Color{ get; set; }
```

Thickness

Summary

Gets or sets the line thickness.

Syntax

```
public int Thickness{ get; set; }
```

LineStyle

Summary

Gets or sets the line style.

Syntax

```
public LineStyle LineStyle{ get; set; }
```

ChartZoomEventArgs

Summary

Provides data for the chart type change event.

Syntax

```
public class ChartZoomEventArgs : Object
```

Members

Name	Type	Summary
Chart (8)	Property	Gets the chart.

Chart

Summary

Gets the chart.

Syntax

```
public Chart Chart{ get; }
```

All classes in cAlgo.API.Collections

Name	Type	Summary
IReadonlyList	Interface	Represents a read only collection of a specified type

IReadonlyList

Summary

Represents a read only collection of a specified type

Syntax

```
public interface IReadonlyList : IEnumerable
```

Members

Name	Type	Summary
Count	Property	The total number of elements contained in the collection
this[int index]	Property	Represents the item contained in the collection at a specific index

Count

Summary

The total number of elements contained in the collection

Syntax

```
public int Count{ get; }
```

this[int index]

Summary

Represents the item contained in the collection at a specific index

Syntax

```
public T this[int index]{ get; }
```

Parameters

Name	Description

Color

Summary

Represents an ARGB (alpha, red, green, blue) color.

Syntax

```
public sealed class Color : Object
```

Members

Name	Type	Summary

A	Property	Gets the alpha component value of the color.
AliceBlue	Property	Gets a system-defined color that has an ARGB value of #FFF0F8FF.
AntiqueWhite	Property	Gets a system-defined color that has an ARGB value of #FFFAEBD7.
Aqua	Property	Gets a system-defined color that has an ARGB value of #FF00FFFF.
Aquamarine	Property	Gets a system-defined color that has an ARGB value of #FF7FFFD4.
Azure	Property	Gets a system-defined color that has an ARGB value of #FFF0FFFF.
B	Property	Gets the blue component value of the color.
Beige	Property	Gets a system-defined color that has an ARGB value of #FFF5F5DC.
Bisque	Property	Gets a system-defined color that has an ARGB value of #FFFFE4C4.
Black	Property	Gets a system-defined color that has an ARGB value of #FF000000.
BlanchedAlmond	Property	Gets a system-defined color that has an ARGB value of #FFFFEBCD.
Blue	Property	Gets a system-defined color that has an ARGB value of #FF0000FF.
BlueViolet	Property	Gets a system-defined color that has an ARGB value of #FF8A2BE2.
Brown	Property	Gets a system-defined color that has an ARGB value of #FFA52A2A.
BurlyWood	Property	Gets a system-defined color that has an ARGB value of #FFDEB887.
CadetBlue	Property	Gets a system-defined color that has an ARGB value of #FF5F9EA0.
Chartreuse	Property	Gets a system-defined color that has an ARGB value of #FF7FFF00.
Chocolate	Property	Gets a system-defined color that has an ARGB value of #FFD2691E.
Coral	Property	Gets a system-defined color that has an ARGB value of #FFFF7F50.
CornflowerBlue	Property	Gets a system-defined color that has an ARGB value of #FF6495ED.
Cornsilk	Property	Gets a system-defined color that has an ARGB value of #FFFFF8DC.
Crimson	Property	Gets a system-defined color that has an ARGB value of #FFDC143C.
Cyan	Property	Gets a system-defined color that has an ARGB value of #FF00FFFF.
DarkBlue	Property	Gets a system-defined color that has an ARGB value of #FF00008B.
DarkCyan	Property	Gets a system-defined color that has an ARGB value of #FF008B8B.
DarkGoldenrod	Property	Gets a system-defined color that has an ARGB value of #FFB8860B.
DarkGray	Property	Gets a system-defined color that has an ARGB value of #FFA9A9A9.
DarkGreen	Property	Gets a system-defined color that has an ARGB value of #FF006400.
DarkKhaki	Property	Gets a system-defined color that has an ARGB value of #FFBDB76B.
DarkMagenta	Property	Gets a system-defined color that has an ARGB value of #FF8B008B.
DarkOliveGreen	Property	Gets a system-defined color that has an ARGB value of #FF556B2F.
DarkOrange	Property	Gets a system-defined color that has an ARGB value of #FFFF8C00.
DarkOrchid	Property	Gets a system-defined color that has an ARGB value of #FF9932CC.
DarkRed	Property	Gets a system-defined color that has an ARGB value of #FF8B0000.

DarkSalmon	Property	Gets a system-defined color that has an ARGB value of #FFE9967A.
DarkSeaGreen	Property	Gets a system-defined color that has an ARGB value of #FF8FBC8F.
DarkSlateBlue	Property	Gets a system-defined color that has an ARGB value of #FF483D8B.
DarkSlateGray	Property	Gets a system-defined color that has an ARGB value of #FF2F4F4F.
DarkTurquoise	Property	Gets a system-defined color that has an ARGB value of #FF00CED1.
DarkViolet	Property	Gets a system-defined color that has an ARGB value of #FF9400D3.
DeepPink	Property	Gets a system-defined color that has an ARGB value of #FFFF1493.
DeepSkyBlue	Property	Gets a system-defined color that has an ARGB value of #FF00BFFF.
DimGray	Property	Gets a system-defined color that has an ARGB value of #FF696969.
DodgerBlue	Property	Gets a system-defined color that has an ARGB value of #FF1E90FF.
Empty	Field	Represents empty color.
Equals	Method	Defines whether the specified object is equal to this instance.
Firebrick	Property	Gets a system-defined color that has an ARGB value of #FFB22222.
FloralWhite	Property	Gets a system-defined color that has an ARGB value of #FFFFFAF0.
ForestGreen	Property	Gets a system-defined color that has an ARGB value of #FF228B22.
FromArgb	Method	Creates a color from alpha, red, green and blue components.
FromHex	Method	Attempts to convert a hex string to a Color.
FromName	Method	Creates a color from the specified name of a predefined color.
Fuchsia	Property	Gets a system-defined color that has an ARGB value of #FFFF00FF.
G	Property	Gets the green component value of the color.
Gainsboro	Property	Gets a system-defined color that has an ARGB value of #FFDCDCDC.
GetHashCode	Method	
GhostWhite	Property	Gets a system-defined color that has an ARGB value of #FFF8F8FF.
Gold	Property	Gets a system-defined color that has an ARGB value of #FFFFD700.
Goldenrod	Property	Gets a system-defined color that has an ARGB value of #FFDAA520.
Gray	Property	Gets a system-defined color that has an ARGB value of #FF808080.
Green	Property	Gets a system-defined color that has an ARGB value of #FF008000.
GreenYellow	Property	Gets a system-defined color that has an ARGB value of #FFADFF2F.
Honeydew	Property	Gets a system-defined color that has an ARGB value of #FFF0FFF0.
HotPink	Property	Gets a system-defined color that has an ARGB value of #FFFF69B4.
IndianRed	Property	Gets a system-defined color that has an ARGB value of #FFCD5C5C.
Indigo	Property	Gets a system-defined color that has an ARGB value of #FF4B0082.
Ivory	Property	Gets a system-defined color that has an ARGB value of #FFFFFFF0.

Khaki	Property	Gets a system-defined color that has an ARGB value of #FFF0E68C.
Lavender	Property	Gets a system-defined color that has an ARGB value of #FFE6E6FA.
LavenderBlush	Property	Gets a system-defined color that has an ARGB value of #FFFFF0F5.
LawnGreen	Property	Gets a system-defined color that has an ARGB value of #FF7CFC00.
LemonChiffon	Property	Gets a system-defined color that has an ARGB value of #FFFFFACD.
LightBlue	Property	Gets a system-defined color that has an ARGB value of #FFADD8E6.
LightCoral	Property	Gets a system-defined color that has an ARGB value of #FFF08080.
LightCyan	Property	Gets a system-defined color that has an ARGB value of #FFE0FFFF.
LightGoldenrodYellow	Property	Gets a system-defined color that has an ARGB value of #FFFAFAD2.
LightGray	Property	Gets a system-defined color that has an ARGB value of #FFD3D3D3.
LightGreen	Property	Gets a system-defined color that has an ARGB value of #FF90EE90.
LightPink	Property	Gets a system-defined color that has an ARGB value of #FFFFB6C1.
LightSalmon	Property	Gets a system-defined color that has an ARGB value of #FFFFA07A.
LightSeaGreen	Property	Gets a system-defined color that has an ARGB value of #FF20B2AA.
LightSkyBlue	Property	Gets a system-defined color that has an ARGB value of #FF87CEFA.
LightSlateGray	Property	Gets a system-defined color that has an ARGB value of #FF778899.
LightSteelBlue	Property	Gets a system-defined color that has an ARGB value of #FFB0C4DE.
LightYellow	Property	Gets a system-defined color that has an ARGB value of #FFFFFFE0.
Lime	Property	Gets a system-defined color that has an ARGB value of #FF00FF00.
LimeGreen	Property	Gets a system-defined color that has an ARGB value of #FF32CD32.
Linen	Property	Gets a system-defined color that has an ARGB value of #FFFAF0E6.
Magenta	Property	Gets a system-defined color that has an ARGB value of #FFFF00FF.
Maroon	Property	Gets a system-defined color that has an ARGB value of #FF800000.
MediumAquamarine	Property	Gets a system-defined color that has an ARGB value of #FF66CDAA.
MediumBlue	Property	Gets a system-defined color that has an ARGB value of #FF0000CD.
MediumOrchid	Property	Gets a system-defined color that has an ARGB value of #FFBA55D3.
MediumPurple	Property	Gets a system-defined color that has an ARGB value of #FF9370DB.
MediumSeaGreen	Property	Gets a system-defined color that has an ARGB value of #FF3CB371.
MediumSlateBlue	Property	Gets a system-defined color that has an ARGB value of #FF7B68EE.
MediumSpringGreen	Property	Gets a system-defined color that has an ARGB value of #FF00FA9A.
MediumTurquoise	Property	Gets a system-defined color that has an ARGB value of #FF48D1CC.
MediumVioletRed	Property	Gets a system-defined color that has an ARGB value of #FFC71585
MidnightBlue	Property	Gets a system-defined color that has an ARGB value of #FF191970.
MintCream	Property	Gets a system-defined color that has an ARGB value of #FFF5FFFA.

MistyRose	Property	Gets a system-defined color that has an ARGB value of #FFFFE4E1.
Moccasin	Property	Gets a system-defined color that has an ARGB value of #FFFFE4B5.
NavajoWhite	Property	Gets a system-defined color that has an ARGB value of #FFFFDEAD.
Navy	Property	Gets a system-defined color that has an ARGB value of #FF000080.
OldLace	Property	Gets a system-defined color that has an ARGB value of #FFFDF5E6.
Olive	Property	Gets a system-defined color that has an ARGB value of #FF808000.
OliveDrab	Property	Gets a system-defined color that has an ARGB value of #FF6B8E23.
Orange	Property	Gets a system-defined color that has an ARGB value of #FFFFA500.
OrangeRed	Property	Gets a system-defined color that has an ARGB value of #FFFF4500.
Orchid	Property	Gets a system-defined color that has an ARGB value of #FFDA70D6.
PaleGoldenrod	Property	Gets a system-defined color that has an ARGB value of #FFEEE8AA.
PaleGreen	Property	Gets a system-defined color that has an ARGB value of #FF98FB98.
PaleTurquoise	Property	Gets a system-defined color that has an ARGB value of #FFAFEEEE.
PaleVioletRed	Property	Gets a system-defined color that has an ARGB value of #FFDB7093.
PapayaWhip	Property	Gets a system-defined color that has an ARGB value of #FFFFEFD5.
PeachPuff	Property	Gets a system-defined color that has an ARGB value of #FFFFDAB9.
Peru	Property	Gets a system-defined color that has an ARGB value of #FFCD853F.
Pink	Property	Gets a system-defined color that has an ARGB value of #FFFFC0CB.
Plum	Property	Gets a system-defined color that has an ARGB value of #FFDDA0DD.
PowderBlue	Property	Gets a system-defined color that has an ARGB value of #FFB0E0E6.
Purple	Property	Gets a system-defined color that has an ARGB value of #FF800080.
R	Property	Gets the red component value of the color.
Red	Property	Gets a system-defined color that has an ARGB value of #FFFF0000.
RosyBrown	Property	Gets a system-defined color that has an ARGB value of #FFBC8F8F.
RoyalBlue	Property	Gets a system-defined color that has an ARGB value of #FF4169E1.
SaddleBrown	Property	Gets a system-defined color that has an ARGB value of #FF8B4513.
Salmon	Property	Gets a system-defined color that has an ARGB value of #FFFA8072.
SandyBrown	Property	Gets a system-defined color that has an ARGB value of #FFF4A460.
SeaGreen	Property	Gets a system-defined color that has an ARGB value of #FF2E8B57.
SeaShell	Property	Gets a system-defined color that has an ARGB value of #FFFFF5EE.
Sienna	Property	Gets a system-defined color that has an ARGB value of #FFA0522D.
Silver	Property	Gets a system-defined color that has an ARGB value of #FFC0C0C0.
SkyBlue	Property	Gets a system-defined color that has an ARGB value of #FF87CEEB.

SlateBlue	Property	Gets a system-defined color that has an ARGB value of #FF6A5ACD.
SlateGray	Property	Gets a system-defined color that has an ARGB value of #FF708090.
Snow	Property	Gets a system-defined color that has an ARGB value of #FFFFFAFA.
SpringGreen	Property	Gets a system-defined color that has an ARGB value of #FF00FF7F.
SteelBlue	Property	Gets a system-defined color that has an ARGB value of #FF4682B4.
Tan	Property	Gets a system-defined color that has an ARGB value of #FFD2B48C.
Teal	Property	Gets a system-defined color that has an ARGB value of #FF008080.
Thistle	Property	Gets a system-defined color that has an ARGB value of #FFD8BFD8.
ToArgb	Method	Get the 32-bit ARGB color value.
ToHexString	Method	Get the hex string representation of the color.
Tomato	Property	Gets a system-defined color that has an ARGB value of #FFFF6347.
ToString	Method	Returns a String that represents this instance.
Transparent	Property	Gets a system-defined color that has an ARGB value of #00FFFFFF.
Turquoise	Property	Gets a system-defined color that has an ARGB value of #FF40E0D0.
Violet	Property	Gets a system-defined color that has an ARGB value of #FFEE82EE.
Wheat	Property	Gets a system-defined color that has an ARGB value of #FFF5DEB3.
White	Property	Gets a system-defined color that has an ARGB value of #FFFFFFFF.
WhiteSmoke	Property	Gets a system-defined color that has an ARGB value of #FFF5F5F5.
Yellow	Property	Gets a system-defined color that has an ARGB value of #FFFFFF00.
YellowGreen	Property	Gets a system-defined color that has an ARGB value of #FF9ACD32.

Example 1

```
var blueColor = Color.Blue;
var greenColor = Color.FromArgb(0, 0, 255, 0);
```

A

Summary

Gets the alpha component value of the color.

Syntax

```
public Byte A{ get; }
```

R

Summary

Gets the red component value of the color.

Syntax

```
public Byte R{ get; }
```

G

Summary

Gets the green component value of the color.

Syntax

```
public Byte G{ get; }
```

B

Summary

Gets the blue component value of the color.

Syntax

```
public Byte B{ get; }
```

Transparent

Summary

Gets a system-defined color that has an ARGB value of #00FFFFFF.

Syntax

```
public static Color Transparent{ get; }
```

AliceBlue

Summary

Gets a system-defined color that has an ARGB value of #FFF0F8FF.

Syntax

```
public static Color AliceBlue{ get; }
```

AntiqueWhite

Summary

Gets a system-defined color that has an ARGB value of #FFFAEBD7.

Syntax

```
public static Color AntiqueWhite{ get; }
```

Aqua

Summary

Gets a system-defined color that has an ARGB value of #FF00FFFF.

Syntax

```

```

```
public static Color Aqua{ get; }
```

Aquamarine

Summary

Gets a system-defined color that has an ARGB value of #FF7FFFD4.

Syntax

```
public static Color Aquamarine{ get; }
```

Azure

Summary

Gets a system-defined color that has an ARGB value of #FFF0FFFF.

Syntax

```
public static Color Azure{ get; }
```

Beige

Summary

Gets a system-defined color that has an ARGB value of #FFF5F5DC.

Syntax

```
public static Color Beige{ get; }
```

Bisque

Summary

Gets a system-defined color that has an ARGB value of #FFFFE4C4.

Syntax

```
public static Color Bisque{ get; }
```

Black

Summary

Gets a system-defined color that has an ARGB value of #FF000000.

Syntax

```
public static Color Black{ get; }
```

BlanchedAlmond

Summary

Gets a system-defined color that has an ARGB value of #FFFFEBCD.

Syntax

```
public static Color BlanchedAlmond{ get; }
```

Blue

Summary

Gets a system-defined color that has an ARGB value of #FF0000FF.

Syntax

```
public static Color Blue{ get; }
```

BlueViolet

Summary

Gets a system-defined color that has an ARGB value of #FF8A2BE2.

Syntax

```
public static Color BlueViolet{ get; }
```

Brown

Summary

Gets a system-defined color that has an ARGB value of #FFA52A2A.

Syntax

```
public static Color Brown{ get; }
```

BurlyWood

Summary

Gets a system-defined color that has an ARGB value of #FFDEB887.

Syntax

```
public static Color BurlyWood{ get; }
```

CadetBlue

Summary

Gets a system-defined color that has an ARGB value of #FF5F9EA0.

Syntax

```
public static Color CadetBlue{ get; }
```

Chartreuse

Summary

Gets a system-defined color that has an ARGB value of #FF7FFF00.

Syntax

```
public static Color Chartreuse{ get; }
```

Chocolate

Summary

Gets a system-defined color that has an ARGB value of #FFD2691E.

Syntax

```
public static Color Chocolate{ get; }
```

Coral

Summary

Gets a system-defined color that has an ARGB value of #FFFF7F50.

Syntax

```
public static Color Coral{ get; }
```

CornflowerBlue

Summary

Gets a system-defined color that has an ARGB value of #FF6495ED.

Syntax

```
public static Color CornflowerBlue{ get; }
```

Cornsilk

Summary

Gets a system-defined color that has an ARGB value of #FFFFF8DC.

Syntax

```
public static Color Cornsilk{ get; }
```

Crimson

Summary

Gets a system-defined color that has an ARGB value of #FFDC143C.

Syntax

```
public static Color Crimson{ get; }
```

Cyan

Summary

Gets a system-defined color that has an ARGB value of #FF00FFFF.

Syntax

```
public static Color Cyan{ get; }
```

DarkBlue

Summary

Gets a system-defined color that has an ARGB value of #FF00008B.

Syntax

```
public static Color DarkBlue{ get; }
```

DarkCyan

Summary

Gets a system-defined color that has an ARGB value of #FF008B8B.

Syntax

```
public static Color DarkCyan{ get; }
```

DarkGoldenrod

Summary

Gets a system-defined color that has an ARGB value of #FFB8860B.

Syntax

```
public static Color DarkGoldenrod{ get; }
```

DarkGray

Summary

Gets a system-defined color that has an ARGB value of #FFA9A9A9.

Syntax

```
public static Color DarkGray{ get; }
```

DarkGreen

Summary

Gets a system-defined color that has an ARGB value of #FF006400.

Syntax

```
public static Color DarkGreen{ get; }
```

DarkKhaki

Summary

Gets a system-defined color that has an ARGB value of #FFBDB76B.

Syntax

```
public static Color DarkKhaki{ get; }
```

DarkMagenta

Summary

Gets a system-defined color that has an ARGB value of #FF8B008B.

Syntax

```
public static Color DarkMagenta{ get; }
```

DarkOliveGreen

Summary

Gets a system-defined color that has an ARGB value of #FF556B2F.

Syntax

```
public static Color DarkOliveGreen{ get; }
```

DarkOrange

Summary

Gets a system-defined color that has an ARGB value of #FFFF8C00.

Syntax

```
public static Color DarkOrange{ get; }
```

DarkOrchid

Summary

Gets a system-defined color that has an ARGB value of #FF9932CC.

Syntax

```
public static Color DarkOrchid{ get; }
```

DarkRed

Summary

Gets a system-defined color that has an ARGB value of #FF8B0000.

Syntax

```
public static Color DarkRed{ get; }
```

DarkSalmon

Summary

Gets a system-defined color that has an ARGB value of #FFE9967A.

Syntax

```
public static Color DarkSalmon{ get; }
```

DarkSeaGreen

Summary

Gets a system-defined color that has an ARGB value of #FF8FBC8F.

Syntax

```
public static Color DarkSeaGreen{ get; }
```

DarkSlateBlue

Summary

Gets a system-defined color that has an ARGB value of #FF483D8B.

Syntax

```
public static Color DarkSlateBlue{ get; }
```

DarkSlateGray

Summary

Gets a system-defined color that has an ARGB value of #FF2F4F4F.

Syntax

```
public static Color DarkSlateGray{ get; }
```

DarkTurquoise

Summary

Gets a system-defined color that has an ARGB value of #FF00CED1.

Syntax

```
public static Color DarkTurquoise{ get; }
```

DarkViolet

Summary

Gets a system-defined color that has an ARGB value of #FF9400D3.

Syntax

```
public static Color DarkViolet{ get; }
```

DeepPink

Summary

Gets a system-defined color that has an ARGB value of #FFFF1493.

Syntax

```
public static Color DeepPink{ get; }
```

DeepSkyBlue

Summary

Gets a system-defined color that has an ARGB value of #FF00BFFF.

Syntax

```
public static Color DeepSkyBlue{ get; }
```

DimGray

Summary

Gets a system-defined color that has an ARGB value of #FF696969.

Syntax

```
```

```
public static Color DimGray{ get; }
```

DodgerBlue

Summary

Gets a system-defined color that has an ARGB value of #FF1E90FF.

Syntax

```
public static Color DodgerBlue{ get; }
```

Firebrick

Summary

Gets a system-defined color that has an ARGB value of #FFB22222.

Syntax

```
public static Color Firebrick{ get; }
```

FloralWhite

Summary

Gets a system-defined color that has an ARGB value of #FFFFFAF0.

Syntax

```
public static Color FloralWhite{ get; }
```

ForestGreen

Summary

Gets a system-defined color that has an ARGB value of #FF228B22.

Syntax

```
public static Color ForestGreen{ get; }
```

Fuchsia

Summary

Gets a system-defined color that has an ARGB value of #FFFF00FF.

Syntax

```
public static Color Fuchsia{ get; }
```

Gainsboro

Summary

Gets a system-defined color that has an ARGB value of #FFDCDCDC.

Syntax

```
public static Color Gainsboro{ get; }
```

GhostWhite

Summary

Gets a system-defined color that has an ARGB value of #FFF8F8FF.

Syntax

```
public static Color GhostWhite{ get; }
```

Gold

Summary

Gets a system-defined color that has an ARGB value of #FFFFD700.

Syntax

```
public static Color Gold{ get; }
```

Goldenrod

Summary

Gets a system-defined color that has an ARGB value of #FFDAA520.

Syntax

```
public static Color Goldenrod{ get; }
```

Gray

Summary

Gets a system-defined color that has an ARGB value of #FF808080.

Syntax

```
public static Color Gray{ get; }
```

Green

Summary

Gets a system-defined color that has an ARGB value of #FF008000.

Syntax

```
public static Color Green{ get; }
```

GreenYellow

Summary

Gets a system-defined color that has an ARGB value of #FFADFF2F.

Syntax

```
public static Color GreenYellow{ get; }
```

Honeydew

Summary

Gets a system-defined color that has an ARGB value of #FFF0FFF0.

Syntax

```
public static Color Honeydew{ get; }
```

HotPink

Summary

Gets a system-defined color that has an ARGB value of #FFFF69B4.

Syntax

```
public static Color HotPink{ get; }
```

IndianRed

Summary

Gets a system-defined color that has an ARGB value of #FFCD5C5C.

Syntax

```
public static Color IndianRed{ get; }
```

Indigo

Summary

Gets a system-defined color that has an ARGB value of #FF4B0082.

Syntax

```
public static Color Indigo{ get; }
```

Ivory

Summary

Gets a system-defined color that has an ARGB value of #FFFFFFF0.

Syntax

```
public static Color Ivory{ get; }
```

Khaki

Summary

Gets a system-defined color that has an ARGB value of #FFF0E68C.

Syntax

```
public static Color Khaki{ get; }
```

Lavender

Summary

Gets a system-defined color that has an ARGB value of #FFE6E6FA.

Syntax

```
public static Color Lavender{ get; }
```

LavenderBlush

Summary

Gets a system-defined color that has an ARGB value of #FFFFF0F5.

Syntax

```
public static Color LavenderBlush{ get; }
```

LawnGreen

Summary

Gets a system-defined color that has an ARGB value of #FF7CFC00.

Syntax

```
public static Color LawnGreen{ get; }
```

LemonChiffon

Summary

Gets a system-defined color that has an ARGB value of #FFFFFACD.

Syntax

```
public static Color LemonChiffon{ get; }
```

LightBlue

Summary

Gets a system-defined color that has an ARGB value of #FFADD8E6.

Syntax

```
public static Color LightBlue{ get; }
```

LightCoral

Summary

Gets a system-defined color that has an ARGB value of #FFF08080.

Syntax

```
public static Color LightCoral{ get; }
```

LightCyan

Summary

Gets a system-defined color that has an ARGB value of #FFE0FFFF.

Syntax

```
public static Color LightCyan{ get; }
```

LightGoldenrodYellow

Summary

Gets a system-defined color that has an ARGB value of #FFFAFAD2.

Syntax

```
public static Color LightGoldenrodYellow{ get; }
```

LightGray

Summary

Gets a system-defined color that has an ARGB value of #FFD3D3D3.

Syntax

```
public static Color LightGray{ get; }
```

LightGreen

Summary

Gets a system-defined color that has an ARGB value of #FF90EE90.

Syntax

```
public static Color LightGreen{ get; }
```

LightPink

Summary

Gets a system-defined color that has an ARGB value of #FFFFB6C1.

Syntax

```
public static Color LightPink{ get; }
```

LightSalmon

Summary

Gets a system-defined color that has an ARGB value of #FFFFA07A.

Syntax

```
public static Color LightSalmon{ get; }
```

LightSeaGreen

Summary

Gets a system-defined color that has an ARGB value of #FF20B2AA.

Syntax

```
public static Color LightSeaGreen{ get; }
```

LightSkyBlue

Summary

Gets a system-defined color that has an ARGB value of #FF87CEFA.

Syntax

```
public static Color LightSkyBlue{ get; }
```

LightSlateGray

Summary

Gets a system-defined color that has an ARGB value of #FF778899.

Syntax

```
public static Color LightSlateGray{ get; }
```

LightSteelBlue

Summary

Gets a system-defined color that has an ARGB value of #FFB0C4DE.

Syntax

```
public static Color LightSteelBlue{ get; }
```

LightYellow

Summary

Gets a system-defined color that has an ARGB value of #FFFFFFE0.

Syntax

```
public static Color LightYellow{ get; }
```

Lime

Summary

Gets a system-defined color that has an ARGB value of #FF00FF00.

Syntax

```
public static Color Lime{ get; }
```

LimeGreen

Summary

Gets a system-defined color that has an ARGB value of #FF32CD32.

Syntax

```
public static Color LimeGreen{ get; }
```

Linen

Summary

Gets a system-defined color that has an ARGB value of #FFFAF0E6.

Syntax

```
```

```
public static Color Linen{ get; }
```

Magenta

Summary

Gets a system-defined color that has an ARGB value of #FFFF00FF.

Syntax

```
public static Color Magenta{ get; }
```

Maroon

Summary

Gets a system-defined color that has an ARGB value of #FF800000.

Syntax

```
public static Color Maroon{ get; }
```

MediumAquamarine

Summary

Gets a system-defined color that has an ARGB value of #FF66CDAA.

Syntax

```
public static Color MediumAquamarine{ get; }
```

MediumBlue

Summary

Gets a system-defined color that has an ARGB value of #FF0000CD.

Syntax

```
public static Color MediumBlue{ get; }
```

MediumOrchid

Summary

Gets a system-defined color that has an ARGB value of #FFBA55D3.

Syntax

```
public static Color MediumOrchid{ get; }
```

MediumPurple

Summary

Gets a system-defined color that has an ARGB value of #FF9370DB.

Syntax

```
public static Color MediumPurple{ get; }
```

MediumSeaGreen

Summary

Gets a system-defined color that has an ARGB value of #FF3CB371.

Syntax

```
public static Color MediumSeaGreen{ get; }
```

MediumSlateBlue

Summary

Gets a system-defined color that has an ARGB value of #FF7B68EE.

Syntax

```
public static Color MediumSlateBlue{ get; }
```

MediumSpringGreen

Summary

Gets a system-defined color that has an ARGB value of #FF00FA9A.

Syntax

```
public static Color MediumSpringGreen{ get; }
```

MediumTurquoise

Summary

Gets a system-defined color that has an ARGB value of #FF48D1CC.

Syntax

```
public static Color MediumTurquoise{ get; }
```

MediumVioletRed

Summary

Gets a system-defined color that has an ARGB value of #FFC71585

Syntax

```
public static Color MediumVioletRed{ get; }
```

MidnightBlue

Summary

Gets a system-defined color that has an ARGB value of #FF191970.

Syntax

```
public static Color MidnightBlue{ get; }
```

MintCream

Summary

Gets a system-defined color that has an ARGB value of #FFF5FFFA.

Syntax

```
public static Color MintCream{ get; }
```

MistyRose

Summary

Gets a system-defined color that has an ARGB value of #FFFFE4E1.

Syntax

```
public static Color MistyRose{ get; }
```

Moccasin

Summary

Gets a system-defined color that has an ARGB value of #FFFFE4B5.

Syntax

```
public static Color Moccasin{ get; }
```

NavajoWhite

Summary

Gets a system-defined color that has an ARGB value of #FFFFDEAD.

Syntax

```
public static Color NavajoWhite{ get; }
```

Navy

Summary

Gets a system-defined color that has an ARGB value of #FF000080.

Syntax

```
public static Color Navy{ get; }
```

OldLace

Summary

Gets a system-defined color that has an ARGB value of #FFFDF5E6.

Syntax

```
public static Color OldLace{ get; }
```

Olive

Summary

Gets a system-defined color that has an ARGB value of #FF808000.

Syntax

```
public static Color Olive{ get; }
```

OliveDrab

Summary

Gets a system-defined color that has an ARGB value of #FF6B8E23.

Syntax

```
public static Color OliveDrab{ get; }
```

Orange

Summary

Gets a system-defined color that has an ARGB value of #FFFFA500.

Syntax

```
public static Color Orange{ get; }
```

OrangeRed

Summary

Gets a system-defined color that has an ARGB value of #FFFF4500.

Syntax

```
public static Color OrangeRed{ get; }
```

Orchid

Summary

Gets a system-defined color that has an ARGB value of #FFDA70D6.

Syntax

```
public static Color Orchid{ get; }
```

PaleGoldenrod

Summary

Gets a system-defined color that has an ARGB value of #FFEEE8AA.

Syntax

```
public static Color PaleGoldenrod{ get; }
```

PaleGreen

Summary

Gets a system-defined color that has an ARGB value of #FF98FB98.

Syntax

```
public static Color PaleGreen{ get; }
```

PaleTurquoise

Summary

Gets a system-defined color that has an ARGB value of #FFAFEEEE.

Syntax

```
public static Color PaleTurquoise{ get; }
```

PaleVioletRed

Summary

Gets a system-defined color that has an ARGB value of #FFDB7093.

Syntax

```
public static Color PaleVioletRed{ get; }
```

PapayaWhip

Summary

Gets a system-defined color that has an ARGB value of #FFFFEFD5.

Syntax

```
public static Color PapayaWhip{ get; }
```

PeachPuff

Summary

Gets a system-defined color that has an ARGB value of #FFFFDAB9.

Syntax

```
public static Color PeachPuff{ get; }
```

Peru

Summary

Gets a system-defined color that has an ARGB value of #FFCD853F.

Syntax

```
public static Color Peru{ get; }
```

Pink

Summary

Gets a system-defined color that has an ARGB value of #FFFFC0CB.

Syntax

```
public static Color Pink{ get; }
```

Plum

Summary

Gets a system-defined color that has an ARGB value of #FFDDA0DD.

Syntax

```
public static Color Plum{ get; }
```

PowderBlue

Summary

Gets a system-defined color that has an ARGB value of #FFB0E0E6.

Syntax

```
public static Color PowderBlue{ get; }
```

Purple

Summary

Gets a system-defined color that has an ARGB value of #FF800080.

Syntax

```
public static Color Purple{ get; }
```

Red

Summary

Gets a system-defined color that has an ARGB value of #FFFF0000.

Syntax

```
public static Color Red{ get; }
```

RosyBrown

Summary

Gets a system-defined color that has an ARGB value of #FFBC8F8F.

Syntax

```
public static Color RosyBrown{ get; }
```

RoyalBlue

Summary

Gets a system-defined color that has an ARGB value of #FF4169E1.

Syntax

```
public static Color RoyalBlue{ get; }
```

SaddleBrown

Summary

Gets a system-defined color that has an ARGB value of #FF8B4513.

Syntax

```
```

```
public static Color SaddleBrown{ get; }
```

Salmon

Summary

Gets a system-defined color that has an ARGB value of #FFFA8072.

Syntax

```
public static Color Salmon{ get; }
```

SandyBrown

Summary

Gets a system-defined color that has an ARGB value of #FFF4A460.

Syntax

```
public static Color SandyBrown{ get; }
```

SeaGreen

Summary

Gets a system-defined color that has an ARGB value of #FF2E8B57.

Syntax

```
public static Color SeaGreen{ get; }
```

SeaShell

Summary

Gets a system-defined color that has an ARGB value of #FFFFF5EE.

Syntax

```
public static Color SeaShell{ get; }
```

Sienna

Summary

Gets a system-defined color that has an ARGB value of #FFA0522D.

Syntax

```
public static Color Sienna{ get; }
```

Silver

Summary

Gets a system-defined color that has an ARGB value of #FFC0C0C0.

Syntax

```
public static Color Silver{ get; }
```

SkyBlue

Summary

Gets a system-defined color that has an ARGB value of #FF87CEEB.

Syntax

```
public static Color SkyBlue{ get; }
```

SlateBlue

Summary

Gets a system-defined color that has an ARGB value of #FF6A5ACD.

Syntax

```
public static Color SlateBlue{ get; }
```

SlateGray

Summary

Gets a system-defined color that has an ARGB value of #FF708090.

Syntax

```
public static Color SlateGray{ get; }
```

Snow

Summary

Gets a system-defined color that has an ARGB value of #FFFFFAFA.

Syntax

```
public static Color Snow{ get; }
```

SpringGreen

Summary

Gets a system-defined color that has an ARGB value of #FF00FF7F.

Syntax

```
public static Color SpringGreen{ get; }
```

SteelBlue

Summary

Gets a system-defined color that has an ARGB value of #FF4682B4.

Syntax

```
public static Color SteelBlue{ get; }
```

Tan

Summary

Gets a system-defined color that has an ARGB value of #FFD2B48C.

Syntax

```
public static Color Tan{ get; }
```

Teal

Summary

Gets a system-defined color that has an ARGB value of #FF008080.

Syntax

```
public static Color Teal{ get; }
```

Thistle

Summary

Gets a system-defined color that has an ARGB value of #FFD8BFD8.

Syntax

```
public static Color Thistle{ get; }
```

Tomato

Summary

Gets a system-defined color that has an ARGB value of #FFFF6347.

Syntax

```
public static Color Tomato{ get; }
```

Turquoise

Summary

Gets a system-defined color that has an ARGB value of #FF40E0D0.

Syntax

```
public static Color Turquoise{ get; }
```

Violet

Summary

Gets a system-defined color that has an ARGB value of #FFEE82EE.

Syntax

```
public static Color Violet{ get; }
```

Wheat

Summary

Gets a system-defined color that has an ARGB value of #FFF5DEB3.

Syntax

```
public static Color Wheat{ get; }
```

White

Summary

Gets a system-defined color that has an ARGB value of #FFFFFFFF.

Syntax

```
public static Color White{ get; }
```

WhiteSmoke

Summary

Gets a system-defined color that has an ARGB value of #FFF5F5F5.

Syntax

```
public static Color WhiteSmoke{ get; }
```

Yellow

Summary

Gets a system-defined color that has an ARGB value of #FFFFFF00.

Syntax

```
public static Color Yellow{ get; }
```

YellowGreen

Summary

Gets a system-defined color that has an ARGB value of #FF9ACD32.

Syntax

```
public static Color YellowGreen{ get; }
```

ToArgb

Summary

Get the 32-bit ARGB color value.

Syntax

```
public int ToArgb()
```

ToHexString

Summary

Get the hex string representation of the color.

Syntax

```
public string ToHexString()
```

ToString

Summary

Returns a String that represents this instance.

Syntax

```
public override string ToString()
```

Equals

Summary

Defines whether the specified object is equal to this instance.

Syntax

```
public override bool Equals(Object obj)
```

Parameters

Name	Description

GetHashCode

Syntax

```
public override int GetHashCode()
```

FromArgb

Summary

Creates a color from alpha, red, green and blue components.

Syntax

```
public static Color FromArgb(int alpha, int red, int green, int blue)
```

```
public static Color FromArgb(int alpha, Color baseColor)
```

```
public static Color FromArgb(int argb)
```

```
public static Color FromArgb(int red, int green, int blue)
```

Parameters

Name	Description

Example 1

```
var greenColor = Color.FromArgb(255, 0, 255, 0);
```

FromArgb

Summary

Creates a color from existing color, but with new specified alpha value.

Syntax

```
public static Color FromArgb(int alpha, int red, int green, int blue)
```

```
public static Color FromArgb(int alpha, Color baseColor)
```

```
public static Color FromArgb(int argb)
```

```
public static Color FromArgb(int red, int green, int blue)
```

Parameters

Name	Description

Example 1

```
var transparentBlue = Color.FromArgb(128, Color.Blue);
```

FromArgb

Summary

Creates a color from a 32-bit ARGB value.

Syntax

```
public static Color FromArgb(int alpha, int red, int green, int blue)
```

```
public static Color FromArgb(int alpha, Color baseColor)
```

```
public static Color FromArgb(int argb)
```

```
```

```
public static Color FromArgb(int red, int green, int blue)
```

Parameters

Name	Description

FromArgb

Summary

Creates a color from red, green and blue values. The alpha value is implicitly 255 (fully opaque).

Syntax

```
public static Color FromArgb(int alpha, int red, int green, int blue)
```

```
public static Color FromArgb(int alpha, Color baseColor)
```

```
public static Color FromArgb(int argb)
```

```
public static Color FromArgb(int red, int green, int blue)
```

Parameters

Name	Description

Example 1

```
var greenColor = Color.FromArgb(0, 255, 0);
```

FromHex

Summary

Attempts to convert a hex string to a Color.

Syntax

```
public static Color FromHex(string hex)
```

Parameters

Name	Description

Example 1

```
var color = Color.FromHex("#808080");
```

FromName

Summary

Creates a color from the specified name of a predefined color.

Syntax

```
public static Color FromName(string name)
```

Parameters

Name	Description

Example 1

```
var greenColor = Color.FromName("Green");
```

Empty

Summary

Represents empty color.

Syntax

```
public static Color Empty
```

DataSeries

Summary

Represents a read only list of values, typically used to represent market price series. The values are accessed with an array-like [] operator.

Syntax

```
public interface DataSeries
```

Members

Name	Type	Summary
Count (2)	Property	Gets the total number of elements contained in the DataSeries.
Last	Method	Access a value in the dataseries certain bars ago
LastValue	Property	Gets the last value of this DataSeries.
this[int index] (2)	Property	Gets the value in the dataseries at the specified position.

Example 1

```
[Parameter]
public DataSeries Source { get; set; }
//...
[Output("Main")]
public IndicatorDataSeries Result{ get; set; }
//...
Result[index] = Source[index] * exp + previousValue * (1 - exp);
//...
Result[index] = (MarketSeries.Close[index] + MarketSeries.Open[index]) / 2;
//...
```

this[int index]

Summary

Gets the value in the dataseries at the specified position.

Syntax

```
public double this[int index]{ get; }
```

Parameters

Name	Description

Example 1

```
//...
[Parameter("Data Source")]
public DataSeries Source { get; set; }
//...
[Output("Main")]
public IndicatorDataSeries Result{ get; set; }
//...
public override void Calculate(int index)
{
    // This is the simple moving average calculation.
    double sum = 0.0;
    for (int i = 0; i <= Periods-1; i++)
    {
        // Source[i] is the item contained in Source at position i
        sum += Source[i];
    }
    Result[index] = sum / Periods;
}
//...
```

LastValue

Summary

Gets the last value of this DataSeries.

Remarks

The last value may represent one of the values of the last bar of the market series, e.g. Open, High, Low and Close. Therefore, take into consideration that on each tick, except the Open price, the rest of the values will most probably change.

Syntax

```
public double LastValue{ get; }
```

Example 1

```
//...
protected override void OnTick()
{
    double lastValue = MarketSeries.Close.LastValue;
    Print("The last value of MarketSeries.Close Series is: {0}",
MarketSeries.Close.LastValue);
    // Property LastValue has an accessor but no setter, i.e. LastValue can be retrieved but
not set.
    // The following code will produce an error
    MarketSeries.Close.LastValue = 100;
}
//...
```

Count

Summary

Gets the total number of elements contained in the DataSeries.

Syntax

```
public int Count{ get; }
```

Example 1

```
protected override void OnTick()
{
    int total = MarketSeries.Close.Count;
    Print("The total elements contained in the MarketSeries.Close Series is: {0}", total);
    int lastIndex = total - 1;
    double lastCloseValue = MarketSeries.Close[lastIndex];
    //Print the last value of the series
    Print("The last value of Close Series is: {0}", lastCloseValue);
}
```

Last

Summary

Access a value in the dataseries certain bars ago

Syntax

```
public double Last(int index)
```

Parameters

Name	Description

Example 1

```
double value = MarketSeries.Close.Last(5);
Print("The close price 5 bars ago was: {0}", value);
```

Example 2

```
double previousOpen = MarketSeries.Open.Last(1);
double previousClose = MarketSeries.Close.Last(1);
Print("Open: {0}, Close: {1}", previousOpen, previousClose);
```

Example 3

```
double currentClose = MarketSeries.Close.Last(0);
Print("Current Close: {0}", currentClose);
```

Error

Summary

Encapsulates an error code.

Syntax

```
public interface Error
```

Members

Name	Type	Summary
Code	Property	The encapsulated error code.
TradeResult	Property	The result of the trade that produced the error

Example 1

```
protected override void OnError(Error error)
{
    // Print the error code
    Print("{0}", error.Code);
}
```

Code

Summary

The encapsulated error code.

Syntax

```
public ErrorCode Code{ get; }
```

Example 1

```
protected override void OnError(Error error)
{
    // stop the robot if there is a volume error
    if (error.Code == ErrorCode.BadVolume)
        Stop();
}
```

TradeResult

Summary

The result of the trade that produced the error

Syntax

```
public TradeResult TradeResult{ get; }
```

Example 1

```
protected override void OnError(Error error)
{
    var result = error.TradeResult;
    Print(result);
}
```

ErrorCode

Summary

Enumeration of standard error codes.

Remarks

Error codes are readable descriptions of the responses returned by the server.

Syntax

```
public sealed enum ErrorCode
```

Members

Name	Type	Summary
BadVolume	Field	The volume value is not valid
Disconnected	Field	The server is disconnected.
EntityNotFound	Field	Position does not exist.
MarketClosed	Field	The market is closed.
NoMoney	Field	There are not enough money in the account to trade with.
TechnicalError	Field	A generic technical error with a trade request.
Timeout	Field	Operation timed out.

Example 1

```
protected override void OnError(Error error)
{
    //  Print the error to the log
    switch (error.Code)
    {
            case ErrorCode.BadVolume: Print("Bad Volume");
            break;
            case ErrorCode.TechnicalError:Print("Technical Error");
            break;
            case ErrorCode.NoMoney: Print("No Money");
            break;
            case ErrorCode.Disconnected: Print("Disconnected");
            break;
            case ErrorCode.MarketClosed: Print("Market Closed");
            break;
    }
}
```

TechnicalError

Summary

A generic technical error with a trade request.

Syntax

```
ErrorCode.TechnicalError
```

Example 1

```
protected override void OnError(Error error)
{
    if (error.Code == ErrorCode.TechnicalError)
    {
        Print("Error. Confirm that the trade command parameters are valid");
    }
}
```

BadVolume

Summary

The volume value is not valid

Syntax

```
ErrorCode.BadVolume
```

Example 1

```
protected override void OnError(Error error)
{
    if (error.Code == ErrorCode.BadVolume)
    {
        Print("Invalid Volume amount");
    }
}
```

NoMoney

Summary

There are not enough money in the account to trade with.

Syntax

```
ErrorCode.NoMoney
```

Example 1

```
protected override void OnError(Error error)
{
    if (error.Code == ErrorCode.NoMoney)
    {
        Print("Not enough money to trade.");
    }
}
```

MarketClosed

Summary

The market is closed.

Syntax

```
ErrorCode.MarketClosed
```

Example 1

```
protected override void OnError(Error error)
{
    if (error.Code == ErrorCode.MarketClosed)
    {
        Print("The market is closed.");
    }
}
```

Disconnected

Summary

The server is disconnected.

Syntax

```
ErrorCode.Disconnected
```

Example 1

```
protected override void OnError(Error error)
{
    if (error.Code == ErrorCode.Disconnected)
    {
        Print("The server is disconnected.");
    }
}
```

EntityNotFound

Summary

Position does not exist.

Syntax

```
ErrorCode.EntityNotFound
```

Example 1

```
protected override void OnError(Error error)
{
    if (error.Code == ErrorCode.EntityNotFound)
    {
        Print("Position not found");
    }
}
```

Timeout

Summary

Operation timed out.

Syntax

```
ErrorCode.Timeout
```

Example 1

```
protected override void OnError(Error error)
{
    if (error.Code == ErrorCode.Timeout)
    {
        Print("Operation timed out");
    }
}
```

FibonacciLevel

Summary

Represents the Fibonacci Level.

Syntax

```
public interface FibonacciLevel
```

Members

Name	Type	Summary
IsVisible	Property	Defines if the level is visible.
PercentLevel	Property	Gets or sets the percent level.

PercentLevel

Summary

Gets or sets the percent level.

Syntax

```
public double PercentLevel{ get; set; }
```

IsVisible

Summary

Defines if the level is visible.

Syntax

```
public bool IsVisible{ get; set; }
```

Functions

Summary

This class contains valuable functions that apply to DataSeries.

Syntax

```
public static sealed class Functions : Object
```

Members

Name	Type	Summary
HasCrossedAbove	Method	Returns true, if dataseries1 has crossed above dataseries2, over the specified Period.
HasCrossedBelow	Method	Returns true, if dataseries1 has crossed below dataseries2, over the specified Period.
IsFalling	Method	Checks if the last value in a dataseries is less than the previous
IsRising	Method	Checks if the last value in a dataseries is greater than the previous.
Maximum	Method	Finds the maximum value in a dataseries for a given period.
Minimum	Method	Finds the minimum of a dataseries for a given period.
Sum	Method	Calculates the sum of a dataseries, over the specified period.

Example 1

```
//...
SimpleMovingAverage sma;
protected override void Initialize()
{
    sma = Indicators.SimpleMovingAverage(source, period);
}
public override void Calculate(int index)
{
    // IsRising returns true if the current value is greater
    // than the previous value in the data series
    if (Functions.IsRising(sma.Result))
    {
        //Do something
    }
    // IsFalling returns true if the current value is less
    // than the previous value in the data series
    else if(Functions.IsFalling(sma.Result))
    {
        // Do something else
```

```
    }
    else // sma is level
    {
        Do something else
    }
    //...
}
```

IsRising

Summary

Checks if the last value in a dataseries is greater than the previous.

Syntax

```
public static bool IsRising(DataSeries series)
```

Parameters

Name	Description

Example 1

```
SimpleMovingAverage sma;
//...
public override void Calculate(int index)
{
    if (Functions.IsRising(sma.Result))
    {
        //Do something
    }
    //May be invoked as an extension method
    if (sma.Result.IsRising())
    {
        //Do something
    }
}
//...
```

IsFalling

Summary

Checks if the last value in a dataseries is less than the previous

Syntax

```
public static bool IsFalling(DataSeries series)
```

Parameters

Name	Description

Example 1

```
SimpleMovingAverage sma;
//...
public override void Calculate(int index)
{
    if (Functions.IsFalling(sma.Result))
    {
        //Do something
    }
    // May also be invoked as an extension method
    if (sma.Result.IsFalling())
    {
        //Do something
    }
}
```

Maximum

Summary

Finds the maximum value in a dataseries for a given period.

Syntax

```
public static double Maximum(DataSeries series, int period)
```

Parameters

Name	Description

Example 1

```
public override void Calculate(int index)
{
    if(Functions.Maximum(sma.Result,20) > MarketSeries.Close[index])
    {
        //Do something
    }
    // May be invoked as an extension method

    if (sma.Result.Maximum(20) > MarketSeries.Close[index])
    {
        //Do something
    }
}
```

Example 2

```
var maxHigh = MarketSeries.High.Maximum(periods);
```

Minimum

Summary

Finds the minimum of a dataseries for a given period.

Syntax

```
public static double Minimum(DataSeries series, int period)
```

Parameters

Name	Description

Example 1

```
public override void Calculate(int index)
{
    if(Functions.Minimum(sma.Result, 20) > MarketSeries.Close[index])
    {
        //Do something
    }
```

```
    // May be invoked as an extension method

    if (sma.Result.Minimum(20) > MarketSeries.Close[index])
    {
        //Do something
    }
}
```

Example 2

```
var minLow = MarketSeries.Low.Minimum(periods);
```

HasCrossedAbove

Summary

Returns true, if dataseries1 has crossed above dataseries2, over the specified Period.

Remarks

HasCrossedAbove will compare the crossing dataseries to the crossed dataseries starting from the current value of the series going back the specified period. If period is zero only the current bar values will be compared. If period is one the current bar values will be compared as well as the previous. e.g. Functions.HasCrossedAbove(sma.Result, MarketSeries.Close, 0) will only compare the current values which are not completed until the close of the bar. It is not uncommon that the function will return true and by the end of the bar the two series will uncross.

Syntax

```
public static bool HasCrossedAbove(DataSeries crossingSeries, DataSeries crossedSeries, int
period)
```

```
public static bool HasCrossedAbove(DataSeries crossingSeries, double value, int period)
```

Parameters

Name	Description

Example 1

```
public override void Calculate(int index)
{
    if(Functions.HasCrossedAbove(sma.Result, MarketSeries.Close, 0))
    {
```

```
        //Do something
    }
    // May be invoked as an extension method as well

    if(sma.Result.HasCrossedAbove(MarketSeries.Close, 0))
    {
        //Do something
    }
}
```

HasCrossedBelow

Summary

Returns true, if dataseries1 has crossed below dataseries2, over the specified Period.

Remarks

HasCrossedBelow will compare the crossing dataseries to the crossed dataseries starting from the current value of the series going back the specified period. If period is zero only the current bar values will be compared. If period is one the current bar values will be compared as well as the previous. e.g. Functions.HasCrossedBelow(sma.Result, MarketSeries.Close, 0) will only compare the current values which are not completed until the close of the bar. It is not uncommon that the function will return true and by the end of the bar the two series will uncross.

Syntax

```
public static bool HasCrossedBelow(DataSeries crossingSeries, DataSeries crossedSeries, int
period)
```

```
public static bool HasCrossedBelow(DataSeries crossingSeries, double value, int period)
```

Parameters

Name	Description

Example 1

```
public override void Calculate(int index)
{
    if(Functions.HasCrossedBelow(sma.Result, MarketSeries.Close,0)
    {
        //Do something
    }
    // May be invoked as an extension method
```

```
    if(sma.Result.HasCrossedBelow(MarketSeries.Close, 0))
    {
        //Do something
    }
}
```

HasCrossedAbove

Summary

Checks if dataseries1 has crossed above value, sometime within the specified period.

Remarks

HasCrossedAbove will compare the crossing dataseries to the crossed dataseries starting from the current value of the series going back the specified period. If period is zero only the current bar values will be compared. If period is one the current bar values will be compared as well as the previous. e.g. Functions.HasCrossedAbove(sma.Result, value, 0) will only compare the current simple moving average series value which is not completed until the close of the bar. It is not uncommon that the function will return true and by the end of the bar the series will uncross.

Syntax

```
public static bool HasCrossedAbove(DataSeries crossingSeries, DataSeries crossedSeries, int
period)
```

```
public static bool HasCrossedAbove(DataSeries crossingSeries, double value, int period)
```

Parameters

Name	Description

Example 1

```
public override void Calculate(int index)
{
    var value = MarketSeries.Close[index - 1];
    if(Functions.HasCrossedAbove(sma.Result, value, 1)
    {
        //Do something
    }
    // May be invoked as an extension method as well

    if(sma.Result.HasCrossedAbove(MarketSeries.Close[index-1], 1))
```

```
    {
        //Do something
    }
}
```

HasCrossedBelow

Summary

Checks if dataseries1 has crossed below the value, sometime within the specified period.

Remarks

HasCrossedBelow compares the crossing dataseries to the value starting from the current value of the series going back the specified period. If period is zero, only the current bar value will be examined. If period is one, the current and previous bar value will be examined. e.g. Functions.HasCrossedAbove(sma.Result, value, 0) will only compare the current simple moving average series value which is not completed until the close of the bar. It is not uncommon that the function will return true and by the end of the bar the series will uncross.

Syntax

```
public static bool HasCrossedBelow(DataSeries crossingSeries, DataSeries crossedSeries, int
period)
```

```
public static bool HasCrossedBelow(DataSeries crossingSeries, double value, int period)
```

Parameters

Name	Description

Example 1

```
public override void Calculate(int index)
{
    if(Functions.HasCrossedBelow(sma.Result, MarketSeries.Close[index], 0)
    {
        //Do something
    }
    // May be invoked as an extension method as well
    if(sma.Result.HasCrossedBelow(MarketSeries.Close[index], 0))
    {
        //Do something
    }
}
```

Sum

Summary

Calculates the sum of a dataseries, over the specified period.

Syntax

```
public static double Sum(DataSeries series, int period)
```

Parameters

Name	Description

Example 1

```
SimpleMovingAverage sma;
//...
public override void Calculate(int index)
{
    //The sum of the simple moving average series of the last 20 bars
    var sumSma = Functions.Sum(sma.Result, 20);
    //...
}
```

GetFitnessArgs

Syntax

```
public interface GetFitnessArgs
```

Members

Name	Type	Summary
AverageTrade	Property	Average profit for all trades
Equity	Property	Represents the equity of the account (balance plus unrealized profit and loss)

History	Property	Collection of all historical trades
LosingTrades	Property	Total number of losing trades
MaxBalanceDrawdown	Property	The maximum amount of balance drawdown in deposit currency.
MaxBalanceDrawdownPercentages	Property	The maximum amount of balance drawdown (%).
MaxEquityDrawdown	Property	The maximum amount of equity drawdown in deposit currency.
MaxEquityDrawdownPercentages	Property	The maximum amount of equity drawdown (%).
NetProfit	Property	The net profit of all trades
PendingOrders	Property	Collection of all pending orders
Positions (2)	Property	Collection of all open positions
ProfitFactor	Property	Profit Factor is the ratio of Total Net Profit divided by Total Net Loss
SharpeRatio	Property	A ratio to measure risk-adjusted performance. The higher the value, the better.
SortinoRatio	Property	The Sortino ratio is an alternative to the Sharpe ratio, using downward deviation in place of standard deviation. The higher the value, the better.
TotalTrades	Property	Total number of trades taken
WinningTrades	Property	Total number of winning trades

History

Summary

Collection of all historical trades

Syntax

```
public History History{ get; }
```

Positions

Summary

Collection of all open positions

Syntax

```

```

```
public Positions Positions{ get; }
```

PendingOrders

Summary

Collection of all pending orders

Syntax

```
public PendingOrders PendingOrders{ get; }
```

Equity

Summary

Represents the equity of the account (balance plus unrealized profit and loss)

Syntax

```
public double Equity{ get; }
```

NetProfit

Summary

The net profit of all trades

Syntax

```
public double NetProfit{ get; }
```

MaxBalanceDrawdownPercentages

Summary

The maximum amount of balance drawdown (%).

Syntax

```
public double MaxBalanceDrawdownPercentages{ get; }
```

MaxEquityDrawdownPercentages

Summary

The maximum amount of equity drawdown (%).

Syntax

```
public double MaxEquityDrawdownPercentages{ get; }
```

MaxBalanceDrawdown

Summary

The maximum amount of balance drawdown in deposit currency.

Syntax

```
public double MaxBalanceDrawdown{ get; }
```

MaxEquityDrawdown

Summary

The maximum amount of equity drawdown in deposit currency.

Syntax

```
public double MaxEquityDrawdown{ get; }
```

WinningTrades

Summary

Total number of winning trades

Syntax

```
public double WinningTrades{ get; }
```

LosingTrades

Summary

Total number of losing trades

Syntax

```
public double LosingTrades{ get; }
```

TotalTrades

Summary

Total number of trades taken

Syntax

```
public double TotalTrades{ get; }
```

AverageTrade

Summary

Average profit for all trades

Syntax

```
public double AverageTrade{ get; }
```

ProfitFactor

Summary

Profit Factor is the ratio of Total Net Profit divided by Total Net Loss

Syntax

```
public double ProfitFactor{ get; }
```

SharpeRatio

Summary

A ratio to measure risk-adjusted performance. The higher the value, the better.

Syntax

```
public double SharpeRatio{ get; }
```

SortinoRatio

Summary

The Sortino ratio is an alternative to the Sharpe ratio, using downward deviation in place of standard deviation. The higher

the value, the better.

Syntax

```
public double SortinoRatio{ get; }
```

HideFromIntellisenceAttribute

Syntax

```
public class HideFromIntellisenceAttribute : Attribute
```

Members

Name	Type	Summary
HideFromIntellisenceAttribute	Method	

HideFromIntellisenceAttribute

Syntax

```
public HideFromIntellisenceAttribute HideFromIntellisenceAttribute()
```

HistoricalTrade

Summary

Represents historical trade

Syntax

```
public interface HistoricalTrade
```

Members

Name	Type	Summary
Balance	Property	Account balance after the Deal was filled
ClosingDealId	Property	Unique deal identifier
ClosingPrice	Property	The execution price of the Closing Deal
ClosingTime	Property	Time of the Closing Deal
Comment (2)	Property	The comment
Commissions	Property	Commission owed
EntryPrice	Property	The VWAP (Volume Weighted Average Price) of the Opening Deals that are closed
EntryTime	Property	Time of the Opening Deal, or the time of the first Opening deal that was closed
GrossProfit	Property	Profit and loss before swaps and commission
Label	Property	The label
NetProfit (2)	Property	Profit and loss including swaps and commissions
Pips	Property	Represents the winning or loosing pips
PositionId	Property	The position's unique identifier.
Quantity	Property	The Quantity (in lots) that was closed by the Closing Deal
Swap	Property	Swap is the overnight interest rate if any, accrued on the position.
SymbolCode	Property	Symbol code of the Historical Trade.
TradeType	Property	The TradeType of the Opening Deal
VolumeInUnits	Property	The Volume that was closed by the Closing Deal

ClosingDealId

Summary

Unique deal identifier

Syntax

```
public int ClosingDealId{ get; }
```

PositionId

Summary

The position's unique identifier.

Syntax

```
public int PositionId{ get; }
```

SymbolCode

Summary

Symbol code of the Historical Trade.

Syntax

```
public string SymbolCode{ get; }
```

TradeType

Summary

The TradeType of the Opening Deal

Syntax

```
public TradeType TradeType{ get; }
```

VolumeInUnits

Summary

The Volume that was closed by the Closing Deal

Syntax

```
public double VolumeInUnits{ get; }
```

EntryTime

Summary

Time of the Opening Deal, or the time of the first Opening deal that was closed

Syntax

```
public DateTime EntryTime{ get; }
```

EntryPrice

Summary

The VWAP (Volume Weighted Average Price) of the Opening Deals that are closed

Syntax

```
public double EntryPrice{ get; }
```

ClosingTime

Summary

Time of the Closing Deal

Syntax

```
public DateTime ClosingTime{ get; }
```

ClosingPrice

Summary

The execution price of the Closing Deal

Syntax

```
public double ClosingPrice{ get; }
```

Label

Summary

The label

Syntax

```
public string Label{ get; }
```

Comment

Summary

The comment

Syntax

```
public string Comment{ get; }
```

Commissions

Summary

Commission owed

Syntax

```
```

```
public double Commissions{ get; }
```

Swap

Summary

Swap is the overnight interest rate if any, accrued on the position.

Syntax

```
public double Swap{ get; }
```

NetProfit

Summary

Profit and loss including swaps and commissions

Syntax

```
public double NetProfit{ get; }
```

GrossProfit

Summary

Profit and loss before swaps and commission

Syntax

```
public double GrossProfit{ get; }
```

Balance

Summary

Account balance after the Deal was filled

Syntax

```
public double Balance{ get; }
```

Pips

Summary

Represents the winning or loosing pips

Syntax

```
public double Pips{ get; }
```

Quantity

Summary

The Quantity (in lots) that was closed by the Closing Deal

Syntax

```
public double Quantity{ get; }
```

History

Summary

Provides access to methods of the historical trades collection

Syntax

```
public interface History : IEnumerable
```

Members

Name	Type	Summary
Count (3)	Property	Total number of historical trades
FindAll	Method	Find all historical trades by the label
FindLast	Method	Find last historical trade by its label
this[int index] (3)	Property	Find a historical trade by index

this[int index]

Summary

Find a historical trade by index

Syntax

```
public HistoricalTrade this[int index]{ get; }
```

Parameters

Name	Description

Count

Summary

Total number of historical trades

Syntax

```
public int Count{ get; }
```

FindLast

Summary

Find last historical trade by its label

Syntax

```
public HistoricalTrade FindLast(string label)
```

```
public HistoricalTrade FindLast(string label, Symbol symbol)
```

```
public HistoricalTrade FindLast(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

FindLast

Summary

Find last historical trade by its label, symbol

Syntax

```
public HistoricalTrade FindLast(string label)
```

```
public HistoricalTrade FindLast(string label, Symbol symbol)
```

```
public HistoricalTrade FindLast(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

FindLast

Summary

Find last historical trade by its label, symbol and trade type

Syntax

```
public HistoricalTrade FindLast(string label)
```

```
public HistoricalTrade FindLast(string label, Symbol symbol)
```

```
public HistoricalTrade FindLast(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

FindAll

Summary

Find all historical trades by the label

Syntax

```
public HistoricalTrade[] FindAll(string label)
```

```
public HistoricalTrade[] FindAll(string label, Symbol symbol)
```

```
public HistoricalTrade[] FindAll(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

FindAll

Summary

Find all historical trades by label and symbol

Syntax

```
public HistoricalTrade[] FindAll(string label)
```

```
public HistoricalTrade[] FindAll(string label, Symbol symbol)
```

```
public HistoricalTrade[] FindAll(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

FindAll

Summary

Find all historical trades by label, symbol and trade type

Syntax

```
public HistoricalTrade[] FindAll(string label)
```

```
public HistoricalTrade[] FindAll(string label, Symbol symbol)
```

```
public HistoricalTrade[] FindAll(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

HorizontalAlignment

Summary

Describes horizontal position related to an anchor point or a parent element

Syntax

```
public sealed enum HorizontalAlignment
```

Members

Name	Type	Summary
Center	Field	Center horizontal alignment.
Left	Field	Left horizontal alignment.
Right	Field	Right horizontal alignment.
Stretch	Field	

Center

Summary

Center horizontal alignment.

Syntax

```
HorizontalAlignment.Center
```

Left

Summary

Left horizontal alignment.

Syntax

```
HorizontalAlignment.Left
```

Right

Summary

Right horizontal alignment.

Syntax

```
HorizontalAlignment.Right
```

Stretch

Syntax

```
HorizontalAlignment.Stretch
```

Indicator

Summary

Base class for Indicators.

Remarks

Contains all necessary market information, provides access to built-in indicators and provides framework for convenient indicators' creation.

Syntax

```
public class Indicator : Algo, IIndicator
```

Members

Name	Type	Summary
Account	Property	Contains information of the current account.
Calculate	Method	Calculate the value(s) of indicator for the given index.
Indicator	Method	Indicator class constructor

IndicatorArea	Property	Defines the area where the indicator is placed.
Initialize	Method	Custom initialization for the Indicator. This method is invoked when an indicator is launched.
IsLastBar	Property	Returns true, if Calculate is invoked for the last bar
ToString (2)	Method	The name of the indicator derived class.

Example 1

```
//...
public override void Calculate(int index)
{
    //This is where we place our indicator's calculation logic.
}
//...
```

Example 2

```
//...
protected override void Initialize()
{
//Place your Initialization logic here
}
//...
```

Example 3

```
private IndicatorDataSeries input;
protected override void Initialize()
{
    input = CreateDataSeries();
}
public override void Calculate(int index)
{
    input[index] = (MarketSeries.Close[index] + MarketSeries.Open[index]) / 2;
}
```

Example 4

```
//...
public override void Calculate(int index)
{
    if (IsRealTime)
    {
        //Place the code-logic that you want to be calculated on incoming live data
    }
```

```
}
//...
```

IsLastBar

Summary

Returns true, if Calculate is invoked for the last bar

Syntax

```
public bool IsLastBar{ get; }
```

Example 1

```
public override void Calculate(int index)
   if (IsLastBar)
   {
       // this is the current (last) index
   }
```

IndicatorArea

Summary

Defines the area where the indicator is placed.

Syntax

```
public IndicatorArea IndicatorArea{ get; }
```

Account

Summary

Contains information of the current account.

Syntax

```
public IAccount Account{ get; }
```

Example 1

```
if (Account.Balance < 10000)
    Print(Account.Balance);
```

Indicator

Summary

Indicator class constructor

Syntax

```
protected Indicator Indicator()
```

Calculate

Summary

Calculate the value(s) of indicator for the given index.

Syntax

```
public void Calculate(int index)
```

Parameters

Name	Description

Example 1

```
//...
```

```
public override void Calculate(int index)
{
    //This is where we place our indicator's calculation logic.
}
//...
```

Initialize

Summary

Custom initialization for the Indicator. This method is invoked when an indicator is launched.

Syntax

```
protected virtual void Initialize()
```

Example 1

```
//...
protected override void Initialize()
{
    //Place your Initialization logic here
}
//...
```

ToString

Summary

The name of the indicator derived class.

Syntax

```
public override string ToString()
```

Example 1

```
private SampleSMA sma;
```

```
//...
sma = Indicators.GetIndicator<SampleSMA>(Source, Period);
Print(sma.ToString());
```

IndicatorArea

Summary

Represents the area where the Indicator is placed.

Syntax

```
public interface IndicatorArea : ChartArea
```

Members

Name	Type	Summary

IndicatorAreaAddedEventArgs

Summary

Provides data for the indicator area adding event.

Syntax

```
public class IndicatorAreaAddedEventArgs : IndicatorAreaEventArgs
```

Members

Name	Type	Summary

IndicatorAreaEventArgs

Summary

The arguments for the indicator area event.

Syntax

```
public class IndicatorAreaEventArgs : Object
```

Members

Name	Type	Summary
Area (3)	Property	Gets the area.
Chart (9)	Property	Gets the chart.

Chart

Summary

Gets the chart.

Syntax

```
public Chart Chart{ get; }
```

Area

Summary

Gets the area.

Syntax

```
public IndicatorArea Area{ get; }
```

IndicatorAreaRemovedEventArgs

Summary

Provides data for the indicator area removing event.

Syntax

```
public class IndicatorAreaRemovedEventArgs : IndicatorAreaEventArgs
```

Members

Name	Type	Summary

IndicatorAttribute

Summary

Indicator Attribute. Applies metadata to enable the indicator plot.

Remarks

To make it effective apply enclosed in square brackets, e.g. [Indicator] before the indicator class declaration. Cannot be ommited.

Syntax

```
public sealed class IndicatorAttribute : Attribute
```

Members

Name	Type	Summary
AccessRights	Property	AccessRights required for Indicator
AutoRescale	Property	Indicates whether this instance automatically rescales the chart or not
IndicatorAttribute	Method	Initializes a new instance of the IndicatorAttribute and sets the name of the indicator.
IsOverlay	Property	Indicates whether this instance is overlayed on the chart or plotted on a separate indicator panel
Name (2)	Property	The name of the Indicator.
ScalePrecision	Property	The price scale precision.
TimeZone	Property	The chart timezone of the displayed indicator

Example 1

```
namespace cAlgo.Indicators
   [Indicator()]
   public class SampleIndicator : Indicator
   {
```

```
        //...
    }
```

Example 2

```
[Indicator("Custom Indicator" )]
public class SampleIndicator : Indicator
```

Example 3

```
[Indicator("IndicatorName", ScalePrecision = 5, IsOverlay = false, TimeZone = TimeZones.UTC)]
public class SampleIndicator : Indicator
```

Name

Summary

The name of the Indicator.

Remarks

The name is displayed to the left of the indicator panel.

Syntax

```
public string Name{ get; }
```

Example 1

```
namespace cAlgo.Indicators
{
    [Indicator("IndicatorName")]
    public class SampleIndicator : Indicator
    {
        //...
    }
}
```

ScalePrecision

Summary

The price scale precision.

Remarks

The number of decimals displayed on the price scale of the indicator panel

Syntax

```
public int ScalePrecision{ get; set; }
```

Example 1

```
namespace cAlgo.Indicators
{
    [Indicator(ScalePrecision = 5)] // The scale precision is 5 decimals.
    public class SampleIndicator : Indicator
    {
        //...
    }
}
```

IsOverlay

Summary

Indicates whether this instance is overlayed on the chart or plotted on a separate indicator panel

Syntax

```
public bool IsOverlay{ get; set; }
```

Example 1

```
[Indicator(IsOverlay = true)] // Plots the Indicator on the chart
public class SampleIndicator : Indicator
```

Example 2

```
[Indicator(IsOverlay = false)] // Plots the Indicator on a separate indicator panel.
public class SampleIndicator : Indicator
```

AutoRescale

Summary

Indicates whether this instance automatically rescales the chart or not

Syntax

```
public bool AutoRescale{ get; set; }
```

Example 1

```
[Indicator(AutoRescale = false)]
public class SampleIndicator : Indicator
```

TimeZone

Summary

The chart timezone of the displayed indicator

Syntax

```
public string TimeZone{ get; set; }
```

Example 1

```
[Indicator(TimeZone = TimeZones.UTC)]
public class SampleIndicator : Indicator
```

AccessRights

Summary

AccessRights required for Indicator

Syntax

```
public AccessRights AccessRights{ get; set; }
```

IndicatorAttribute

Summary

Initializes a new instance of the IndicatorAttribute and sets the name of the indicator.

Remarks

To make it effective apply enclosed in square brackets, e.g. [Indicator("IndicatorName")] before the indicator class declaration. The name is displayed on the top left of the indicator panel.

Syntax

```
public IndicatorAttribute IndicatorAttribute(string name)
```

```
public IndicatorAttribute IndicatorAttribute()
```

Parameters

Name	Description

Example 1

```
[Indicator("IndicatorName")]
public class SampleIndicator : Indicator
```

IndicatorAttribute

Summary

Initializes a new instance of the IndicatorAttribute

Remarks

To make it effective apply enclosed in square brackets, e.g. [Indicator] before the indicator class declaration. The name is displayed on the top left of the indicator panel.

Syntax

```
public IndicatorAttribute IndicatorAttribute(string name)
```

```
public IndicatorAttribute IndicatorAttribute()
```

Example 1

```
namespace cAlgo.Indicators
{
    [Indicator()]
    public class SampleIndicator : Indicator
    {
        //...
    }
}
```

IndicatorDataSeries

Summary

Represents a mutable array of values. An extension of DataSeries used to represent indicator values.

Syntax

```
public interface IndicatorDataSeries : DataSeries
```

Members

Name	Type	Summary
this[int index] (4)	Property	Gets or sets the value at the specified index.

Example 1

```
//This will be the output result of your indicator
[Output("Result", Color = Colors.Orange)]
public IndicatorDataSeries Result { get; set; }
```

Example 2

```
//  The following example is the calculation of the simple moving average
//  of the median price
[Output("Result")]
public IndicatorDataSeries Result { get; set; }
private IndicatorDataSeries _dataSeries;
private SimpleMovingAverage _simpleMovingAverage;
protected override void Initialize()
{
    _dataSeries = CreateDataSeries();
    _simpleMovingAverage = Indicators.SimpleMovingAverage(_dataSeries, 14);
}
public override void Calculate(int index)
{
    _dataSeries[index] = (MarketSeries.High[index] + MarketSeries.Low[index])/2;
    Result[index] = _simpleMovingAverage.Result[index];
}
```

this[int index]

Summary

Gets or sets the value at the specified index.

Syntax

```
public double this[int index]{ get; set; }
```

Parameters

Name	Description

Example 1

```
//  The following example is the calculation of the median price
[Output("Result")]
public IndicatorDataSeries Result { get; set; }
```

```
private IndicatorDataSeries _dataSeries;
protected override void Initialize()
{
    _dataSeries = CreateDataSeries();
}
public override void Calculate(int index)
{

    _dataSeries[index] = (MarketSeries.High[index] + MarketSeries.Low[index])/2;

    // Get the value of _dataSeries at index
    // and set the value of Result at index
    Result[index] = _dataSeries[index];
}
```

All classes in cAlgo.API.Indicators

Name	Type	Summary
AcceleratorOscillator	Interface	Identifies possible trend reversals
AccumulativeSwingIndex	Interface	A variation on Wilder's swing index which plots an accumulation of the swing index value of each candlestick or bar.
Aroon	Interface	An indicator for identifying trends in a currency pair, as well as for gauging the probability of a trend reversal.
AverageTrueRange	Interface	Average true range. An indicator providing the degree of price volatility.
AwesomeOscillator	Interface	Displays market momentum as a histogram
BollingerBands	Interface	Bollinger Bands are used to confirm signals. The bands indicate overbought and oversold levels relative to a moving average.
ChaikinMoneyFlow	Interface	Chaikin Money Flow measures the amount of Money Flow Volume over a specific period. The resulting indicator fluctuates above/below the zero line.
ChaikinVolatility	Interface	Calculates a Chaikin Volatility Indicator
CommodityChannelIndex	Interface	Calculates a Commodity Channel Index
DetrendedPriceOscillator	Interface	Calculates the Detrended Price Oscillator Indicator
DirectionalMovementSystem	Interface	Welles Wilder's Directional Movement Indicator calculation
DonchianChannel	Interface	The Donchian channel is a volatility indicator forming a channel between the highest high and the lowest low of the chosen period.
EaseOfMovement	Interface	Ease of Movement is a volume based oscillator that measures the "ease" of price movement.
ExponentialMovingAverage	Interface	The exponential moving average of the price data source over a period of time.

FractalChaosBands	Interface	The Fractal Chaos Bands indicator attempts to determine whether or not the market is trending.
HighMinusLow	Interface	Difference between MarketSeries.High and MarketSeries.Low calculation for each index
HistoricalVolatility	Interface	The measured price fluctuation over a specified time period.
IchimokuKinkoHyo	Interface	Ichimoku Kinko Hyo Indicator.
KeltnerChannels	Interface	Keltner Channels are volatility-based envelopes set above and below an exponential moving average
LinearRegressionForecast	Interface	Linear Regression Forecast is one of the indicators calculated by the Linear Regression approach.
LinearRegressionIntercept	Interface	Linear Regression Intercept is one of the indicators calculated by the Linear Regression approach.
LinearRegressionRSquared	Interface	Linear Regression R Squared is used to confirm the strength of the market trend
LinearRegressionSlope	Interface	The calculation of Linear Regression Slope Indicator
MacdCrossOver	Interface	Calculates a MACD (moving average convergence/divergence) Indicator
MacdHistogram	Interface	The calculation of the MACD Histogram
MassIndex	Interface	The calculation of Mass Index Indicator
MedianPrice	Interface	A Median Price is an average of one period’s high and low values.
MomentumOscillator	Interface	The calculation of a momentum oscillator
MoneyFlowIndex	Interface	The Money Flow Index is an oscillator that calculates buying and selling pressure using typical price and volume. It oscillates between zero and one hundred. It is typically used to identify trend reversals and price extremes.
MovingAverage	Interface	Moving Average Indicator calculation
NegativeVolumeIndex	Interface	Dysart's Negative Volume Index assumes that the smart money is active on days when volume decreases and the not-so-smart money is active on days when volume increases (measured by the Positive Volume Index).
OnBalanceVolume	Interface	On Balance Volume measures buying and selling pressure as a cumulative indicator that adds volume on up days and subtracts volume on down days.
ParabolicSAR	Interface	The calculation of Parabolic SAR Indicator
PositiveVolumeIndex	Interface	The positive volume index measures the trend of the stock prices for days when volume increases from previous day's volume.
PriceOscillator	Interface	The Price Oscillator calculates the spread between a short-period moving average and a long-period moving average.
PriceROC	Interface	The Price ROC calculates the percentage change between the most recent price and the n-periods of past price.
PriceVolumeTrend	Interface	Price and Volume Trend is a variation of On Balance Volume, used to determine the strength of trends and warn of reversals.
		Developed by Mel Widner, Rainbow Oscillator is based on multiple

RainbowOscillator	Interface	moving averages and helps to identify trends and provides overbought/oversold levels.
RelativeStrengthIndex	Interface	The RSI (Wilder) is momentum oscillator, measuring the velocity and magnitude of directional price movements.
SimpleMovingAverage	Interface	The simple moving average is an average of price within n previous periods.
StandardDeviation	Interface	Standard Deviation measures the market volatility with a commonly used statisctical function.
StochasticOscillator	Interface	The Stochastic Oscillator is a momentum indicator that aims to show price reversals by comparing the closing price to the price range.
SwingIndex	Interface	Developed by Welles Wilder, the Swing Index compares current Open, high, Low and Close prices to find of current and previous periods to find "real" price.
TimeSeriesMovingAverage	Interface	A Time Series Moving Average is moving average based on linear regression forecast.
TradeVolumeIndex	Interface	Trade Volume Index measures the amount of money flowing in and out of an asset.
TriangularMovingAverage	Interface	The Triangular Moving Average is a moving average that gives more weith to values located in the middle of aggregated period.
Trix	Interface	TRIX was developed by Jack Huton. It is a momentum oscillator that will help you filter unimportant price movement.
TrueRange	Interface	The Average True Range is a measure of market volatility developed by Wilder.
TypicalPrice	Interface	A Typical Price is an average of high, low and close values for a single period.
UltimateOscillator	Interface	The Ultimate Oscillator is a technical analysis oscillator based on a notion of buying or selling "pressure".
VerticalHorizontalFilter	Interface	Vertical Horizontal Filter determines whether a price is going through a congestion phase or a trending phase.
Vidya	Interface	Volatility Index Dynamic Average (VIDYA) is a smoothing (moving average) based on dynamically changing periods.
VolumeOscillator	Interface	The Volume Oscillator identifies trends in volume using a two moving average system. A strong trend is signaled when it is positive. Falling volume indicates trend weakness.
VolumeROC	Interface	The Volume Rate of Change indicator measures the Rate Of Change of the tick volume.
WeightedClose	Interface	Weighted Close is an average of high, low and close prices where close has greater weight.
WeightedMovingAverage	Interface	The Weighted Moving Average is a moving average that gives more weith to the latest values.
WellesWilderSmoothing	Interface	The Welles Wilder's Smoothing indicator is an exponential moving average, but it has different alpha ration. As a result it responds to price changes slower.
WilliamsAccumulationDistribution	Interface	William's Accumulation Distribution is an oscillator that can identify if the market is driven by buyers (accumulation) or by sellers (distribution)

WilliamsPctR	Interface	Williams %R is an effective momentum oscillator and was described by Larry Williams for the first time in 1973.

AcceleratorOscillator

Summary

Identifies possible trend reversals

Syntax

```
public interface AcceleratorOscillator
```

Members

Name	Type	Summary
Result	Property	AcceleratorOscillator calculation result

Example 1

```
protected override void Initialize()
{
    acceleratorOscillator = Indicators.AcceleratorOscillator();
}
```

Result

Summary

AcceleratorOscillator calculation result

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
protected override void OnBar()
{
```

```
    var lastValue = acceleratorOscillator.Result.LastValue;
}
```

AccumulativeSwingIndex

Summary

A variation on Wilder's swing index which plots an accumulation of the swing index value of each candlestick or bar.

Remarks

The accumulative swing index is used to gain a longer-term picture than the Wilder's swing index. When the accumulative swing index is positive, the long-term trend is up. When the accumulative swing index is negative, it signals a downwards long-term trend.

Syntax

```
public interface AccumulativeSwingIndex
```

Members

Name	Type	Summary
Result (2)	Property	The time series of AccumulativeSwingIndex.

Example 1

```
using cAlgo.API;
using cAlgo.API.Indicators;
namespace cAlgo.Indicator
{
    [Indicator]
    public class AccumSwingIndexReferenceExample:Indicator
    {
        private AccumulativeSwingIndex _accumulativeSwingIndex;
        [Parameter("Limit Move", DefaultValue = 12)]
        public int LimitMove { get; set; }
        [Output("Main")]
        public IndicatorDataSeries Result { get; set; }
        protected override void Initialize()
        {
            _accumulativeSwingIndex = Indicators.AccumulativeSwingIndex(LimitMove);
        }
        public override void Calculate(int index)
        {
            // Display Result of Indicator
```

```
            Result[index] = _accumulativeSwingIndex.Result[index];
        }
    }
}
```

Result

Summary

The time series of AccumulativeSwingIndex.

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
//...
private AccumulativeSwingIndex _accumulativeSwingIndex;
//...
[Parameter("Limit Move", DefaultValue = 12)]
public int LimitMove { get; set; }

//...
protected override void OnStart()
{
    _accumulativeSwingIndex = Indicators.AccumulativeSwingIndex(LimitMove);
}
protected override void OnBar()
{
    // Print to log
    Print("The Current Accumulative Swing Index is: {0}",
_accumulativeSwingIndex.Result.LastValue);
}
//...
```

Aroon

Summary

An indicator for identifying trends in a currency pair, as well as for gauging the probability of a trend reversal.

Remarks

The indicator fluctuates between 0 and 100, with values above 80 signalling an upward trend, and values below 20 signalling a downward trend.

Syntax

```
public interface Aroon
```

Members

Name	Type	Summary
Down	Property	Aroon Down
Up	Property	Aroon Up

Example 1

```
using cAlgo.API;
using cAlgo.API.Indicators;
namespace cAlgo.Indicators
{
    [Indicator]
    public class AroonReferenceExample:Indicator
    {
        private Aroon _aroon;
        [Parameter("Periods", DefaultValue = 25)]
        public int Periods { get; set; }
        [Output("Up")]
        public IndicatorDataSeries ResultAroonUp { get; set; }
        [Output("Down")]
        public IndicatorDataSeries ResultAroonDown { get; set; }
        protected override void Initialize()
        {
            _aroon = Indicators.Aroon(Periods);
        }
        public override void Calculate(int index)
        {
            ResultAroonUp[index] = _aroon.Up[index];
            ResultAroonDown[index] = _aroon.Down[index];
        }
    }
}
```

Up

Summary

Aroon Up

Syntax

```
public IndicatorDataSeries Up{ get; }
```

Example 1

```
//...
[Parameter("Periods", DefaultValue = 25)]
public int Periods { get; set; }
//...
private Aroon _aroon;
//...
protected override void OnStart()
{
    _aroon = Indicators.Aroon(Periods);
}
protected override void OnBar()
{
    Print("Current Aroon Up Value is: {0}", _aroon.Up.LastValue);
}
//...
```

Down

Summary

Aroon Down

Syntax

```
public IndicatorDataSeries Down{ get; }
```

Example 1

```
//...
[Parameter("Periods", DefaultValue = 25)]
public int Periods { get; set; }
//...
```

```
private Aroon _aroon;
//...
protected override void OnStart()
{
    _aroon = Indicators.Aroon(Periods);
}
protected override void OnBar()
{
    Print("Current Aroon Down Value is: {0}", _aroon.Down.LastValue);
}
//...
```

AverageTrueRange

Summary

Average true range. An indicator providing the degree of price volatility.

Remarks

Average true range is a volatility indicator originally developed by J. Welles Wilder. The indicator provides the degree of price volatility. The average true range is an N-day (exponential) moving average of the true range values. Wilder recommended a 14-period smoothing.

Syntax

```
public interface AverageTrueRange
```

Members

Name	Type	Summary
Result (3)	Property	The resulting data series of Average True Range Indicator instance

Example 1

```
private AverageTrueRange averageTrueRange;
[Parameter(DefaultValue = 14)]
public int Periods { get; set; }
[Parameter(DefaultValue = 0.002)]
public double ATRValue { get; set; }
[Parameter("MA Type", DefaultValue = MovingAverageType.Exponential)]
public MovingAverageType MAType { get; set; }
protected override void OnStart()
{
    averageTrueRange = Indicators.AverageTrueRange(Periods, MAType);
```

```
}
protected override void OnTick()
{
    // if the 14 day Average True Range is higher than 0.002
    if(averageTrueRange.Result.LastValue >= ATRValue)
    {
        // Do Something
    }
}
```

Result

Summary

The resulting data series of Average True Range Indicator instance

Syntax

```
public IndicatorDataSeries Result{ get; set; }
```

Example 1

```
public override void Calculate(int index)
{
    // Plot the Average True Range of period 14
    Result[index] = averageTrueRange.Result[index];
}
```

AwesomeOscillator

Summary

Displays market momentum as a histogram

Syntax

```
public interface AwesomeOscillator
```

Members

Name	Type	Summary
Result (4)	Property	AwesomeOscillator calculation result

Example 1

```
protected override void Initialize()
{
    awesomeOscillator = Indicators.AwesomeOscillator();
}
```

Result

Summary

AwesomeOscillator calculation result

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
protected override void OnBar()
{
    var lastValue = awesomeOscillator.Result.LastValue;
}
```

BollingerBands

Summary

Bollinger Bands are used to confirm signals. The bands indicate overbought and oversold levels relative to a moving average.

Remarks

Bollinger bands widen in volatile market periods, and contract during less volatile periods. Tightening of the bands is often used a signal that there will shortly be a sharp increase in market volatility.

Syntax

```
public interface BollingerBands
```

Members

Name	Type	Summary
Bottom	Property	Lower Bollinger Band.
Main	Property	Moving Average (Middle Bollinger Band).
Top	Property	Upper Bollinger Band.

Example 1

```
//...
[Robot]
public class SampleRobot : Robot
//...
[Parameter("Source")]
public DataSeries Source { get; set; }
[Parameter("BandPeriods", DefaultValue = 14)]
public int BandPeriod { get; set; }
[Parameter("Std", DefaultValue = 14)]
public int std { get; set; }
[Parameter("MAType")]
public MovingAverageType MAType { get; set; }
//...
private BollingerBands boll;
//...
protected override void OnStart()
{
    boll = Indicators.BollingerBands(Source,BandPeriod,std,MAType);
}
protected override void OnBar()
{
    Print("Current Main Bollinger Band's price is: {0}", boll.Main.LastValue);
    Print("Current Bottom Bollinger Band's price is: {0}", boll.Bottom.LastValue);
    Print("Current Top Bollinger Band's price is: {0}", boll.Top.LastValue);
}
//...
```

Main

Summary

Moving Average (Middle Bollinger Band).

Syntax

```
public IndicatorDataSeries Main{ get; }
```

Example 1

```
//...
[Robot]
public class SampleRobot : Robot
//...
[Parameter("Source")]
public DataSeries Source { get; set; }
[Parameter("BandPeriods", DefaultValue = 14)]
public int BandPeriod { get; set; }
[Parameter("Std", DefaultValue = 14)]
public int std { get; set; }
[Parameter("MAType")]
public MovingAverageType MAType { get; set; }
//...
private BollingerBands boll;
//...
protected override void OnStart()
{
    boll = Indicators.BollingerBands(Source,BandPeriod,std,MAType);
}
protected override void OnBar()
{
    Print("Current Main Bollinger Band's price is: {0}", boll.Main.LastValue);
}
//...
```

Top

Summary

Upper Bollinger Band.

Syntax

```
public IndicatorDataSeries Top{ get; }
```

Example 1

```
//...
[Robot]
public class SampleRobot : Robot
//...
[Parameter("Source")]
public DataSeries Source { get; set; }
[Parameter("BandPeriods", DefaultValue = 14)]
public int BandPeriod { get; set; }
[Parameter("Std", DefaultValue = 14)]
public int std { get; set; }
[Parameter("MAType")]
public MovingAverageType MAType { get; set; }
//...
private BollingerBands boll;
//...
protected override void OnStart()
{
    boll = Indicators.BollingerBands(Source,BandPeriod,std,MAType);
}
protected override void OnBar()
{
    Print("Current Top Bollinger Band's price is: {0}", boll.Top.LastValue);
}
//...
```

Bottom

Summary

Lower Bollinger Band.

Syntax

```
public IndicatorDataSeries Bottom{ get; }
```

Example 1

```
//...
[Parameter("Source")]
public DataSeries Source { get; set; }
[Parameter("BandPeriods", DefaultValue = 14)]
public int BandPeriod { get; set; }
[Parameter("Std", DefaultValue = 14)]
public int std { get; set; }
[Parameter("MAType")]
```

```
public MovingAverageType MAType { get; set; }
//...
private BollingerBands boll;
//...
protected override void OnStart()
{
    boll = Indicators.BollingerBands(Source,BandPeriod,std,MAType);
}
protected override void OnBar()
{
    Print("Current Bottom Bollinger Band's price is: {0}", boll.Bottom.LastValue);
}
//...
```

ChaikinMoneyFlow

Summary

Chaikin Money Flow measures the amount of Money Flow Volume over a specific period. The resulting indicator fluctuates above/below the zero line.

Syntax

```
public interface ChaikinMoneyFlow
```

Members

Name	Type	Summary
Result (5)	Property	The time series of ChaikinMoneyFlow Indicator

Example 1

```
private ChaikinMoneyFlow _chaikinMoneyFlow;
[Parameter("Period", DefaultValue = 21)]
public int Period { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
   _chaikinMoneyFlow = Indicators.ChaikinMoneyFlow(Period);
}
public override void Calculate(int index)
{
   // Display Result of Indicator
```

```
    Result[index] = _chaikinMoneyFlow.Result[index];
}
```

Result

Summary

The time series of ChaikinMoneyFlow Indicator

Syntax

```
public IndicatorDataSeries Result{ get; set; }
```

Example 1

```
private ChaikinMoneyFlow _chaikinMoneyFlow;
[Parameter("Period", DefaultValue = 21)]
public int Period { get; set; }
protected override void OnStart()
{
    _chaikinMoneyFlow = Indicators.ChaikinMoneyFlow(Period);
}
protected override void OnBar()
{
    var currentValue = _chaikinMoneyFlow.Result.LastValue;
    //...
}
```

ChaikinVolatility

Summary

Calculates a Chaikin Volatility Indicator

Remarks

The Chaikin Volatility's main purpose is to confirm price trends and to forecast price reversals.

Syntax

```
public interface ChaikinVolatility
```

Members

Name	Type	Summary
Result (6)	Property	Chaikin Volatility Result Series.

Example 1

```
private ChaikinVolatility chaikinVolatility;

[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
    chaikinVolatility = Indicators.ChaikinVolatility(14, 10, MovingAverageType.Simple);
}
public override void Calculate(int index)
{
    // Display Result of Indicator
    Result[index] = _chaikinVolatility.Result[index];

}
```

Result

Summary

Chaikin Volatility Result Series.

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private ChaikinVolatility _chaikinVolatility;
protected override void OnStart()
{
    _chaikinVolatility = Indicators.ChaikinVolatility(Periods, _roc, MaType);
}
protected override void OnBar()
{
    // Print to log
```

```
    Print("The Current Chaikin Volatility Value is: {0}",
            _chaikinVolatility.Result.LastValue);
}
```

CommodityChannelIndex

Summary

Calculates a Commodity Channel Index

Remarks

The Commodity Channel Index is used to determine overbought and oversold conditions relating to a symbol. The Commodity Channel Index can be used to forecast changes in price direction.

Syntax

```
public interface CommodityChannelIndex
```

Members

Name	Type	Summary
Result (7)	Property	Commodity Channel Index Result Series.

Example 1

```
using cAlgo.API;
using cAlgo.API.Indicators;
namespace cAlgo.Indicator
{
    [Indicator]
    public class CommodityChannelIndexReferenceExample:Indicator
    {
        private CommodityChannelIndex _commodityChannelIndex;
        [Parameter("Periods", DefaultValue = 14)]
        public int Periods { get; set; }

        [Output("Main")]
        public IndicatorDataSeries Result { get; set; }
        protected override void Initialize()
        {
            _commodityChannelIndex = Indicators.CommodityChannelIndex(Periods);
        }
        public override void Calculate(int index)
        {
```

```
            // Display Result of Indicator
            Result[index] = _commodityChannelIndex.Result[index];
        }
    }
}
```

Result

Summary

Commodity Channel Index Result Series.

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
//...
private CommodityChannelIndex _commodityChannelIndex;

//...
protected override void OnStart()
{
    _commodityChannelIndex = Indicators.CommodityChannelIndex(Periods);
}
protected override void OnBar()
{
    // Print to log
    Print("The Current Commodity Channel Index is: {0}",
_commodityChannelIndex.Result.LastValue);
}
//...
```

DetrendedPriceOscillator

Summary

Calculates the Detrended Price Oscillator Indicator

Remarks

The Detrended Price Oscillator eliminates trends in prices, showing only absolute changes in price movement.

Syntax

```
public interface DetrendedPriceOscillator
```

Members

Name	Type	Summary
Result (8)	Property	The resulting time series of Detrended Price Oscillator calculation

Example 1

```
private DetrendedPriceOscillator _detrendedPriceOscillator;

[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
    _detrendedPriceOscillator = Indicators.DetrendedPriceOscillator(Source, Periods, MaType);
}
public override void Calculate(int index)
{
    // Display Result of Indicator
    Result[index] = _detrendedPriceOscillator.Result[index];
}
```

Result

Summary

The resulting time series of Detrended Price Oscillator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private _detrendedPriceOscillator _dpoFast;
private _detrendedPriceOscillator _dpoSlow;
protected override void OnStart()
```

```
{
    _dpoFast = Indicators.DetrendedPriceOscillator(Source, PeriodFast, MaType);
    _dpoSlow = Indicators.DetrendedPriceOscillator(Source, PeriodSlow, MaType);
}
protected override void OnBar()
{
    if(_dpoFast.Result.Count < 1)
        return;
    int currentIndex = _dpoFast.Result.Count - 1;
    int prevIndex = currentIndex - 1;
    if (_dpoFast.Result[prevIndex] > _dpoSlow.Result[prevIndex])
    {
        //Do something
    }
}
```

DirectionalMovementSystem

Summary

Welles Wilder's Directional Movement Indicator calculation

Remarks

Welles Wilder's Directional Movement System uses three indicators to determine whether the market is trending, and in which direction, and sends trading signals accordingly. A buy signal occurs when +DI line crosses above -DI line. A sell signal occurs when -DI line crosses below +DI line.

Syntax

```
public interface DirectionalMovementSystem
```

Members

Name	Type	Summary
ADX	Property	The Average Directional Movement Index (ADX) indicates whether the market is trending or ranging.
DIMinus	Property	The Negative Direction Indicator (-DI) indicates downward trend movement;
DIPlus	Property	The Positive Direction Indicator (+DI) indicates upward trend movement;

Example 1

```
//...
[Indicator(IsOverlay = true)]
```

```
public class SampleADX : Indicator
{
    private DirectionalMovementSystem _dms;
    private double _dIplus;
    private double _dIminus;
    [Parameter("ADX Period", DefaultValue = 14)]
    public int Period { get; set; }
    [Output("Buy", PlotType = PlotType.Points, Color = Colors.Green, Thickness = 4)]
    public IndicatorDataSeries Buy { get; set; }
    [Output("Sell", PlotType = PlotType.Points, Color = Colors.Red, Thickness = 4)]
    public IndicatorDataSeries Sell { get; set; }
    protected override void Initialize()
    {
        _dms = Indicators.DirectionalMovementSystem(Period);
    }
    public override void Calculate(int index)
    {
        _dIplus = _dms.DIPlus[index];
        _dIminus = _dms.DIMinus[index];
        if (_dIminus > _dIplus)
        {
            Sell[index] = MarketSeries.Close[index] + Symbol.PointSize*100;
        }
        else
        {
            Buy[index] = MarketSeries.Close[index] - Symbol.PointSize*100;
        }
    }
}
//...
```

ADX

Summary

The Average Directional Movement Index (ADX) indicates whether the market is trending or ranging.

Syntax

```
public IndicatorDataSeries ADX{ get; }
```

Example 1

```
//...
[Robot]
public class SampleRobot : Robot
```

```
//...
private DirectionalMovementSystem _dms;
protected override void Initialize()
{
    _dms = Indicators.DirectionalMovementSystem(Period);
}
//...
protected override void OnBar()
{
    Print("The Current Average Directional Movement Index is: {0}", _dms.ADX.LastValue);
}
//...
```

DIPlus

Summary

The Positive Direction Indicator (+DI) indicates upward trend movement;

Syntax

```
public IndicatorDataSeries DIPlus{ get; }
```

Example 1

```
//...
[Robot]
public class SampleRobot : Robot
//...
private DirectionalMovementSystem _dms;
protected override void Initialize()
{
    _dms = Indicators.DirectionalMovementSystem(Period);
}
//...
protected override void OnBar()
{
    Print("The Current Positive Direction Indicator (+DI) is: {0}", _dms.DIPlus.LastValue);
}
//...
```

DIMinus

Summary

The Negative Direction Indicator (-DI) indicates downward trend movement;

Syntax

```
public IndicatorDataSeries DIMinus{ get; }
```

Example 1

```
//...
[Robot]
public class SampleRobot : Robot
//...
private DirectionalMovementSystem _dms;
protected override void Initialize()
{
    _dms = Indicators.DirectionalMovementSystem(Period);
}
//...
protected override void OnBar()
{
    Print("The Current Negative Direction Indicator (-DI) is: {0}", _dms.DIMinus.LastValue);
}
//...
```

DonchianChannel

Summary

The Donchian channel is a volatility indicator forming a channel between the highest high and the lowest low of the chosen period.

Remarks

The Donchian channel is mainly used for providing entry signals. A long is established when the price closes above the Donchian Channel. Conversely, if it closes below, then a short is established.

Syntax

```
public interface DonchianChannel
```

Members

Name	Type	Summary
Bottom (2)	Property	The lowest low of the period
Middle	Property	The middle of the highest high and the lowest low of the period
Top (2)	Property	The highest high of the period

Example 1

```
//...
private DonchianChannel donchian;
//...
protected override void OnStart()
{
    donchian = Indicators.DonchianChannel(Period);
}
protected override void OnBar()
{
    Print("Top Value = {0}", donchian.Top.LastValue);
    Print("Middle Value = {0}", donchian.Middle.LastValue);
    Print("Bottom Value = {0}", donchian.Bottom.LastValue);
    //...
}
```

Top

Summary

The highest high of the period

Syntax

```
public IndicatorDataSeries Top{ get; set; }
```

Example 1

```
//...
private DonchianChannel donchian;
//...
Print("Top Value = {0}", donchian.Top.LastValue);
```

Middle

Summary

The middle of the highest high and the lowest low of the period

Syntax

```
public IndicatorDataSeries Middle{ get; set; }
```

Example 1

```
//...
private DonchianChannel donchian;
//...
Print("Middle Value = {0}", donchian.Middle.LastValue);
```

Bottom

Summary

The lowest low of the period

Syntax

```
public IndicatorDataSeries Bottom{ get; set; }
```

Example 1

```
//...
private DonchianChannel donchian;
//...
Print("Bottom Value = {0}", donchian.Bottom.LastValue);
```

EaseOfMovement

Summary

Ease of Movement is a volume based oscillator that measures the "ease" of price movement.

Remarks

It quantifies the price/volume relationship. When the oscillator is close to zero it signifies that prices will not move easy. Conversely, prices are advancing or declining with relative ease when the oscillator is positive or negative away from zero.

Syntax

```
public interface EaseOfMovement
```

Members

Name	Type	Summary
Result (9)	Property	The time series of EaseOfMovement Indicator

Example 1

```
private EaseOfMovement _easeOfMovement;
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
[Parameter("MA Type", DefaultValue = MovingAverageType.Simple)]
public MovingAverageType MAType { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
   _easeOfMovement = Indicators.EaseOfMovement(Period, MAType);
}
public override void Calculate(int index)
{
   // Display Result of Indicator
   Result[index] = _easeOfMovement.Result[index];
}
```

Result

Summary

The time series of EaseOfMovement Indicator

Syntax

```
public IndicatorDataSeries Result{ get; set; }
```

Example 1

```
private EaseOfMovement _easeOfMovement;
[Parameter("Period", DefaultValue = 21)]
public int Period { get; set; }
[Parameter("MA Type", DefaultValue = MovingAverageType.Simple)]
public MovingAverageType MAType { get; set; }
protected override void OnStart()
{
    _easeOfMovement = Indicators.EaseOfMovement(Period, MAType);
}
protected override void OnBar()
{
    var currentValue = _easeOfMovement.Result.LastValue;
    //...
}
```

ExponentialMovingAverage

Summary

The exponential moving average of the price data source over a period of time.

Remarks

The exponential moving average is similar to the simple moving average, but applies more weight to more recent data. The weighting for each older price data decreases exponentially. Therefore the exponential moving average reacts faster to latest price changes than the simple moving average.

Syntax

```
public interface ExponentialMovingAverage : MovingAverage, IIndicator
```

Members

Name	Type	Summary

Example 1

```
[Indicator]
public class EmaExample : Indicator
```

```
{
    private ExponentialMovingAverage _emaFast;
    private ExponentialMovingAverage _emaSlow;
    [Parameter("Data Source")]
    public DataSeries Price { get; set; }
    [Parameter("Slow Periods", DefaultValue = 10)]
    public int SlowPeriods { get; set; }
    [Parameter("Fast Periods", DefaultValue = 5)]
    public int FastPeriods { get; set; }
    protected override void Initialize()
    {
        // initialize new instances of ExponentialMovingAverage Indicator class
        _emaFast = Indicators.ExponentialMovingAverage(Price, FastPeriods);
        // _emaSlow is the exponential moving average of the emaFast
        _emaSlow = Indicators.ExponentialMovingAverage(_emaFast.Result, SlowPeriods);
    }
    public override void Calculate(int index)
    {
        // If the index is less than SlowPeriods don't calculate
        if(index <= SlowPeriods)
        {
            return;
        }
        if(_emaFast.Result.HasCrossedAbove(_emaSlow.Result,0))
        {
            // Print the index at which the fast ema crossed the slow ema
            Print("Fast EMA Has Crossed Above at index = {0}", index);
        }
    }
}
```

FractalChaosBands

Summary

The Fractal Chaos Bands indicator attempts to determine whether or not the market is trending.

Remarks

When the market is trending, the bands will have a slope, and if the market is not trending or choppy, the bands will flatten out. The flatter the bands, the stronger the signal that the market is choppy. The more steep the band slopes, the stronger the signal that the market trending or stable.

Syntax

```
public interface FractalChaosBands
```

Members

Name	Type	Summary
High	Property	The high limit of the chaos band.
Low	Property	The low limit of the chaos band.

Example 1

```
private FractalChaosBands _fractalChaosBands;
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
protected override void Initialize()
{
    _fractalChaosBands = Indicators.FractalChaosBands(Period);
}
```

High

Summary

The high limit of the chaos band.

Syntax

```
public IndicatorDataSeries High{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    Print("Fractal Chaos Bands High = {0}", _fractalChaosBands.High[index]);
}
```

Low

Summary

The low limit of the chaos band.

Syntax

```
public IndicatorDataSeries Low{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    Print("Fractal Chaos Bands Low = {0}", _fractalChaosBands.Low[index]);
}
```

HighMinusLow

Summary

Difference between MarketSeries.High and MarketSeries.Low calculation for each index

Remarks

This volatility indicator works by calculating the difference between the high and the low of each trendbar. The larger the difference between high and low, the more volatile the market during that period.

Syntax

```
public interface HighMinusLow
```

Members

Name	Type	Summary
Result (10)	Property	The resulting time series of the calculation

Example 1

```
using cAlgo.API;
using cAlgo.API.Indicators;
namespace cAlgo.Indicators
{
    [Indicator]
    public class Example : Indicator
    {
        private HighMinusLow _highMinusLow;
        protected override void Initialize()
```

```
        {
            _highMinusLow = Indicators.HighMinusLow();
        }
        public override void Calculate(int index)
        {
            // same as MarketSeries.High[index] - MarketSeries.Low[index];
            Print("High minus Low result = {0}", _highMinusLow.Result[index]);
        }
    }
}
```

Result

Summary

The resulting time series of the calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
Print("High minus Low result = {0}", _highMinusLow.Result[index]);
```

HistoricalVolatility

Summary

The measured price fluctuation over a specified time period.

Remarks

The higher the values of the indicator, the more volatile an instrument is.

Syntax

```
public interface HistoricalVolatility
```

Members

Name	Type	Summary
Result (11)	Property	The result of the HistoricalVolatility Indicator

Example 1

```
private HistoricalVolatility historicalVolatility;
private const int BarHistory = 252;
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
protected override void Initialize()
{
    historicalVolatility = Indicators.HistoricalVolatility
        (MarketSeries.Close, Period, BarHistory);
}
```

Result

Summary

The result of the HistoricalVolatility Indicator

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    Print("Historical Volatility = {0}",
         _historicalVolatility.Result[index]);
    }
```

IchimokuKinkoHyo

Summary

Ichimoku Kinko Hyo Indicator.

Remarks

Ichimoku is a moving average based trend identification system. It contains more data points than standard candlestick charts and thus provides a clearer picture of potential price action.

Syntax

```
public interface IchimokuKinkoHyo
```

Members

Name	Type	Summary
ChikouSpan	Property	It is used as a support-resistance aid.
KijunSen	Property	This is a confirmation line, a support-resistance line, and can be used as a trailing stop line.
SenkouSpanA	Property	Leading span 1, this line forms one edge of the kumo, or cloud. If the price is above the Senkou span, the top line serves as the first support level while the bottom line serves as the second support level.
SenkouSpanB	Property	Leading span 2, this line forms the other edge of the kumo.
TenkanSen	Property	It is primarily used as a signal line and a minor support-resistance line.

Example 1

```
//...
private IchimokuKinkoHyo ichimokuKinkoHyo;
//...
protected override void OnStart()
{
    ichimokuKinkoHyo = Indicators.IchimokuKinkoHyo
        (tenkanSenPeriods, kijunSenPeriods, senkouSpanBPeriods);
}
protected override void OnBar()
{
    Print("ChikouSpan Value = {0}", ichimokuKinkoHyo.ChikouSpan.LastValue);
    Print("KijunSen Value = {0}", ichimokuKinkoHyo.KijunSen.LastValue);
    Print("SenkouSpanA Value = {0}", ichimokuKinkoHyo.SenkouSpanA.LastValue);
    Print("SenkouSpanB Value = {0}", ichimokuKinkoHyo.SenkouSpanB.LastValue);
    Print("TenkanSen Value = {0}", ichimokuKinkoHyo.TenkanSen.LastValue);
    //...
}
```

KijunSen

Summary

This is a confirmation line, a support-resistance line, and can be used as a trailing stop line.

Syntax

```
public IndicatorDataSeries KijunSen{ get; set; }
```

Example 1

```
Print("KijunSen Value = {0}", ichimokuKinkoHyo.KijunSen.LastValue);
```

TenkanSen

Summary

It is primarily used as a signal line and a minor support-resistance line.

Syntax

```
public IndicatorDataSeries TenkanSen{ get; set; }
```

Example 1

```
Print("TenkanSen Value = {0}", ichimokuKinkoHyo.TenkanSen.LastValue);
```

ChikouSpan

Summary

It is used as a support-resistance aid.

Syntax

```
public IndicatorDataSeries ChikouSpan{ get; set; }
```

Example 1

```
Print("ChikouSpan Value = {0}", ichimokuKinkoHyo.ChikouSpan.LastValue);
```

SenkouSpanA

Summary

Leading span 1, this line forms one edge of the kumo, or cloud. If the price is above the Senkou span, the top line serves as the first support level while the bottom line serves as the second support level.

Syntax

```
public IndicatorDataSeries SenkouSpanA{ get; set; }
```

Example 1

```
Print("SenkouSpanA Value = {0}", ichimokuKinkoHyo.SenkouSpanA.LastValue);
```

SenkouSpanB

Summary

Leading span 2, this line forms the other edge of the kumo.

Syntax

```
public IndicatorDataSeries SenkouSpanB{ get; set; }
```

Example 1

```
Print("SenkouSpanB Value = {0}", ichimokuKinkoHyo.SenkouSpanB.LastValue);
```

KeltnerChannels

Summary

Keltner Channels are volatility-based envelopes set above and below an exponential moving average

Syntax

```
public interface KeltnerChannels
```

Members

Name	Type	Summary
Bottom (3)	Property	Moving Average - ATR * BandDistance
Main (2)	Property	Moving Average Line
Top (3)	Property	Moving Average + ATR * BandDistance

Main

Summary

Moving Average Line

Syntax

```
public IndicatorDataSeries Main{ get; }
```

Top

Summary

Moving Average + ATR * BandDistance

Syntax

```
public IndicatorDataSeries Top{ get; }
```

Bottom

Summary

Moving Average - ATR * BandDistance

Syntax

```
public IndicatorDataSeries Bottom{ get; }
```

LinearRegressionForecast

Summary

Linear Regression Forecast is one of the indicators calculated by the Linear Regression approach.

Remarks

The Linear Regression Forecast is used for identifying trends and trend direction, and shows the statistical trend of a financial instrument over a specified time period. The calculation uses a Linear Regression Line.

Syntax

```
public interface LinearRegressionForecast
```

Members

Name	Type	Summary
Result (12)	Property	The Result Series of the Linear Regression Forecast Indicator

Example 1

```
private LinearRegressionForecast _linearRegressionForecast;
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
protected override void Initialize()
{
    // initialize a new instance of LinearRegressionForecastIndicator class
    _linearRegressionForecast = Indicators.LinearRegressionForecast(MarketSeries.Close,
Period);
}
```

Result

Summary

The Result Series of the Linear Regression Forecast Indicator

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    // Print the current result of the Linear Regression Forecast to the log
    Print("Linear Regression Forecast at the current index is = {0}",
_linearRegressionForecast.Result[index]);
}
```

LinearRegressionIntercept

Summary

Linear Regression Intercept is one of the indicators calculated by the Linear Regression approach.

Remarks

Linear regression is a statistical tool used to predict the future from past data.

Syntax

```
public interface LinearRegressionIntercept
```

Members

Name	Type	Summary
Result (13)	Property	The Result Series of the Linear Regression Intercept Indicator

Example 1

```

```

```
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
protected override void OnStart()
{
    // initialize a new instance of LinearRegressionIntercept indicator class
    _linearRegressionIntercept = Indicators.LinearRegressionIntercept(MarketSeries.Close,
Period);
}
```

Result

Summary

The Result Series of the Linear Regression Intercept Indicator

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    // Result of _linearRegressionIntercept at the current index
    double result = _linearRegressionIntercept.Result[index];
    // Print the current result to the log
    Print("Linear Regression Intercept at the current index is = {0}", result);
}
```

LinearRegressionRSquared

Summary

Linear Regression R Squared is used to confirm the strength of the market trend

Remarks

A higher value of R-Squared means that the stronger the trend.

Syntax

```
```

```
public interface LinearRegressionRSquared
```

Members

Name	Type	Summary
Result (14)	Property	The Result Series of the LinearRegressionRSquared Indicator

Example 1

```
private LinearRegressionRSquared rSquared;
protected override void OnStart()
{
    // initialize rSquared indicator
    rSquared = Indicators.LinearRegressionRSquared(Source, Period);
}
protected override void OnTick()
{
    Print("{0}", rSquared.Result.LastValue);
}
```

Result

Summary

The Result Series of the LinearRegressionRSquared Indicator

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private LinearRegressionRSquared rSquared;
protected override void OnStart()
{
    // initialize rSquared indicator
    rSquared = Indicators.LinearRegressionRSquared(MarketSeries.Close, 9);
}
protected override void OnTick()
{
    // Print the last value of rSquared indicator to the log
    Print("The current value of R Squared is {0}", rSquared.Result.LastValue);
}
```

LinearRegressionSlope

Summary

The calculation of Linear Regression Slope Indicator

Remarks

Linear Regression Slope refers to the slope of the Least Squares Line. This slope represents how prices change per unit of time.

Syntax

```
public interface LinearRegressionSlope
```

Members

Name	Type	Summary
Result (15)	Property	The resulting time series of the calculation of LinearRegressionSlope Indicator

Example 1

```
private LinearRegressionSlope _lrSlope;
protected override void Initialize()
{
    _lrSlope = Indicators.LinearRegressionSlope(MarketSeries.Close, 14);
}
```

Result

Summary

The resulting time series of the calculation of LinearRegressionSlope Indicator

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    double lr = _lrSlope.Result[index];
}
```

MacdCrossOver

Summary

Calculates a MACD (moving average convergence/divergence) Indicator

Remarks

MACD (moving average convergence/divergence) is used to spot changes in the strength, direction, momentum, and duration of a trend.

Syntax

```
public interface MacdCrossOver
```

Members

Name	Type	Summary
Histogram	Property	The Histogram (bar graph)
MACD	Property	The main MACD line (blue line)
Signal	Property	The Signal line of MACD (red line)

Example 1

```
//...
private MacdCrossOver _macdCrossOver;
[Output("MACD")]
public IndicatorDataSeries Macd { get; set; }
protected override void Initialize()
{
    _macdCrossOver = Indicators.MacdCrossOver(LongCycle, ShortCycle, Period);
    //...
}
public override void Calculate(int index)
{
    Macd[index] = _macdCrossOver.MACD[index];
```

```
    //...
}
```

Histogram

Summary

The Histogram (bar graph)

Remarks

Histogram of MACD: difference between the blue and red lines

Syntax

```
public IndicatorDataSeries Histogram{ get; }
```

Example 1

```
//...
private MacdCrossOver _macdCrossOver;
[Output("Histogram")]
public IndicatorDataSeries Histogram { get; set; }
protected override void Initialize()
{
    _macdCrossOver = Indicators.MacdCrossOver(LongCycle, ShortCycle, Period);
}
public override void Calculate(int index)
{
    Histogram[index] = _macdCrossOver.Histogram[index];
    //...
}
```

MACD

Summary

The main MACD line (blue line)

Remarks

MACD line: difference between the 12 and 26 days EMAs

Syntax

```
public IndicatorDataSeries MACD{ get; }
```

Example 1

```
//...
private MacdCrossOver _macdCrossOver;
[Output("MACD")]
public IndicatorDataSeries Macd { get; set; }
protected override void Initialize()
{
    _macdCrossOver = Indicators.MacdCrossOver(LongCycle, ShortCycle, Period);
}
public override void Calculate(int index)
{
    Macd[index] = _macdCrossOver.MACD[index];
    //...
}
```

Signal

Summary

The Signal line of MACD (red line)

Remarks

Signal: 9 day EMA of the blue line

Syntax

```
public IndicatorDataSeries Signal{ get; }
```

Example 1

```
//...
private MacdCrossOver _macdCrossOver;
[Output("Signal")]
public IndicatorDataSeries Signal { get; set; }
protected override void Initialize()
{
```

```
    _macdCrossOver = Indicators.MacdCrossOver(LongCycle, ShortCycle, Period);
}
public override void Calculate(int index)
{
    Signal[index] = _macdCrossOver.Signal[index];
    //...
}
```

MacdHistogram

Summary

The calculation of the MACD Histogram

Remarks

MACD (moving average convergence/divergence) is used to spot changes in the strength, direction, momentum, and duration of a trend.

Syntax

```
public interface MacdHistogram
```

Members

Name	Type	Summary
Histogram (2)	Property	Histogram (bar graph) The difference between the short and long cycles
Signal (2)	Property	Signal (red line) The exponential moving average of the macd histogram

Example 1

```
//...
private MacdHistogram macd;
//...
macd = Indicators.MacdHistogram(LongCycle, ShortCycle, Period);
//...
```

Histogram

Summary

Histogram (bar graph) The difference between the short and long cycles

Syntax

```
public IndicatorDataSeries Histogram{ get; }
```

Example 1

```
//...
private MacdHistogram macd;
macd = Indicators.MacdHistogram(LongCycle, ShortCycle, Period);
//...
public override void Calculate(int index)
{
    double macdHistogramResult = macd.Histogram[index];
    //...
}
```

Signal

Summary

Signal (red line) The exponential moving average of the macd histogram

Syntax

```
public IndicatorDataSeries Signal{ get; }
```

Example 1

```
//...
private MacdHistogram macd;
macd = Indicators.MacdHistogram(LongCycle, ShortCycle, Period);
//...
public override void Calculate(int index)
{
    double macdSignalResult = macd.Signal[index];
    //...
}
```

MassIndex

Summary

The calculation of Mass Index Indicator

Remarks

The Mass Index can be a great tool to identify future price reversal. It is expected for a reversal to occur when Mass index is rising.

Syntax

```
public interface MassIndex
```

Members

Name	Type	Summary
Result (16)	Property	The resulting series of the calculation of the Mass Index

Example 1

```
private MassIndex _massIndex;
protected override void Initialize()
{
    _massIndex = Indicators.MassIndex(14);
}
public override void Calculate(int index)
{
    double massIndex = _massIndex.Result[index];
}
```

Result

Summary

The resulting series of the calculation of the Mass Index

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    double massIndex = _massIndex.Result[index];
}
```

MedianPrice

Summary

A Median Price is an average of one period’s high and low values.

Remarks

A Median Price is often used as a component for calculating other indicators.

Syntax

```
public interface MedianPrice
```

Members

Name	Type	Summary
Result (17)	Property	The resulting series of the calculation of Median Price

Example 1

```
private MedianPrice _price;
protected override void Initialize()
{
    _price = Indicators.MedianPrice();
}
public override void Calculate(int index)
{
    double price = _price.Result[index];
}
```

Result

Summary

The resulting series of the calculation of Median Price

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private MedianPrice _price;
protected override void Initialize()
{
    _price = Indicators.MedianPrice();
}
public override void Calculate(int index)
{
    double price = _price.Result[index];
}
```

MomentumOscillator

Summary

The calculation of a momentum oscillator

Remarks

Momentum measures the rate of price change over time and provides a leading indicator of changes in trend. It gives signals before price action happens. The momentum oscillator is unbounded i.e. there is no maximum or minimum value. It is calculated as the closing price now minus the closing price n periods ago.

Syntax

```
public interface MomentumOscillator
```

Members

Name	Type	Summary
Result (18)	Property	The resulting series of the momentum oscillator calculation

Example 1

```
private MomentumOscillator _momentum;
protected override void Initialize()
{
    _momentum = Indicators.MomentumOscillator(MarketSeries.Close, 14);
}
public override void Calculate(int index)
{
    double momentum = _momentum.Result[index];
}
```

Result

Summary

The resulting series of the momentum oscillator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    double momentum = _momentum.Result[index];
}
```

MoneyFlowIndex

Summary

The Money Flow Index is an oscillator that calculates buying and selling pressure using typical price and volume. It oscillates between zero and one hundred. It is typically used to identify trend reversals and price extremes.

Syntax

```
public interface MoneyFlowIndex
```

Members

Name	Type	Summary
Result (19)	Property	The time series of MoneyFlowIndex Indicator

Example 1

```
private MoneyFlowIndex _moneyFlow;
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
   _moneyFlow = Indicators.MoneyFlowIndex(Period);
}
public override void Calculate(int index)
{
   // Display Result of Indicator
   Result[index] = _moneyFlow.Result[index];
}
```

Result

Summary

The time series of MoneyFlowIndex Indicator

Syntax

```
public IndicatorDataSeries Result{ get; set; }
```

Example 1

```
private MoneyFlowIndex _moneyFlow;
[Parameter("Period", DefaultValue = 21)]
public int Period { get; set; }
protected override void OnStart()
{
    _moneyFlow = Indicators.MoneyFlowIndex(Period);
}
protected override void OnBar()
{
```

```
    var currentValue = _moneyFlow.Result.LastValue;
    //...
}
```

MovingAverage

Summary

Moving Average Indicator calculation

Remarks

Used to smooth the price data to form a trend following indicator

Syntax

```
public interface MovingAverage : IIndicator
```

Members

Name	Type	Summary
Result (20)	Property	The resulting time series of the calculation

Example 1

```
private MovingAverage ma;
protected override void Initialize()
{
    ma = Indicators.MovingAverage(Source, MAPeriods, MAType);
}
//...
public override void Calculate(int index)
{
    MA[index] = ma.Result[index];
    //...
}
```

Result

Summary

The resulting time series of the calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
//...
[Output]
public IndicatorDataSeries Result { get; set; }
private MovingAverage ma;
protected override void Initialize()
{
    ma = Indicators.MovingAverage(Source, MAPeriods, MAType);
}
public override void Calculate(int index)
{
    Result[index] = ma.Result[index];
    //...
}
```

NegativeVolumeIndex

Summary

Dysart's Negative Volume Index assumes that the smart money is active on days when volume decreases and the not-so-smart money is active on days when volume increases (measured by the Positive Volume Index).

Syntax

```
public interface NegativeVolumeIndex
```

Members

Name	Type	Summary
Result (21)	Property	The time series of NegativeVolumeIndex Indicator

Example 1

```
private NegativeVolumeIndex _negativeVolume;
[Parameter]
```

```
public DataSeries Source { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
   _negativeVolume = Indicators.NegativeVolumeIndex(Source);
}
public override void Calculate(int index)
{
   // Display Result of Indicator
   Result[index] = _negativeVolume.Result[index];
}
```

Result

Summary

The time series of NegativeVolumeIndex Indicator

Syntax

```
public IndicatorDataSeries Result{ get; set; }
```

Example 1

```
private NegativeVolumeIndex _negativeVolume;
[Parameter]
public DataSeries Source { get; set; }
protected override void OnStart()
{
    _negativeVolume = Indicators.NegativeVolumeIndex(Source);
}
protected override void OnBar()
{
    var currentValue = _negativeVolume.Result.LastValue;
    //...
}
```

OnBalanceVolume

Summary

On Balance Volume measures buying and selling pressure as a cumulative indicator that adds volume on up days and subtracts volume on down days.

Syntax

```
public interface OnBalanceVolume
```

Members

Name	Type	Summary
Result (22)	Property	The time series of OnBalanceVolume Indicator

Example 1

```
private OnBalanceVolume _onBalanceVolume;
[Parameter]
public DataSeries Source { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
   _onBalanceVolume = Indicators.OnBalanceVolume(Source);
}
public override void Calculate(int index)
{
   // Display Result of Indicator
   Result[index] = _onBalanceVolume.Result[index];
}
```

Result

Summary

The time series of OnBalanceVolume Indicator

Syntax

```
public IndicatorDataSeries Result{ get; set; }
```

Example 1

```
private OnBalanceVolume _onBalanceVolume;
```

```
[Parameter]
public DataSeries Source { get; set; }
protected override void OnStart()
{
    _onBalanceVolume = Indicators.OnBalanceVolume(Source);
}
protected override void OnBar()
{
    var currentValue = _onBalanceVolume.Result.LastValue;
    //...
}
```

ParabolicSAR

Summary

The calculation of Parabolic SAR Indicator

Remarks

Developed by Welles Wilder, SAR stands for stop and reverse and is based on a concept similar to time decay, unless a security can continue to generate more profits over time, it should be liquidated. SAR trails prices as the trend extends over time, being below prices when they are increasing and above prices when they are decreasing. In this view, the indicator stops and reverses when the price trend reverses and breaks above or below the indicator. The indicator generally works well in trending markets, but not during non-trending, sideways phases. Therefore, Wilder recommended establishing the strength and direction of the trend first through the use of other indicators and then using the Parabolic SAR to trade that trend.The indicator is below prices when prices are rising and above prices when prices are falling. In this regard, the indicator stops and reverses when the price trend reverses and breaks above or below the indicator.

Syntax

```
public interface ParabolicSAR
```

Members

Name	Type	Summary
Result (23)	Property	The resulting series of Parabolic SAR Indicator

Example 1

```
private ParabolicSAR _parabolic;
protected override void Initialize()
{
    _parabolic = Indicators.ParabolicSAR(minaf, maxaf);
}
public override void Calculate(int index)
```

```
{
    double parabolic = _parabolic.Result[index];
}
```

Result

Summary

The resulting series of Parabolic SAR Indicator

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private ParabolicSAR _parabolic;
protected override void Initialize()
{
    _parabolic = Indicators.ParabolicSAR(minaf, maxaf);
}
public override void Calculate(int index)
{
    double parabolic = _parabolic.Result[index];
}
```

PositiveVolumeIndex

Summary

The positive volume index measures the trend of the stock prices for days when volume increases from previous day's volume.

Remarks

Assumes that the smart money is active on days when volume decreases (measured by the Negative Volume Index) and the not-so-smart money is active on days when volume increases.

Syntax

```
public interface PositiveVolumeIndex
```

Members

Name	Type	Summary
Result (24)	Property	The time series result of the PositiveVolumeIndex Indicator instance

Example 1

```
private PositiveVolumeIndex _positiveVolume;
[Parameter]
public DataSeries Source { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
   _positiveVolume = Indicators.PositiveVolumeIndex(Source);
}
public override void Calculate(int index)
{
   // Display Result of Indicator
   Result[index] = _positiveVolume.Result[index];
}
```

Result

Summary

The time series result of the PositiveVolumeIndex Indicator instance

Syntax

```
public IndicatorDataSeries Result{ get; set; }
```

Example 1

```
protected override void OnBar()
{
    var currentValue = _positiveVolume.Result.LastValue;
    //...
}
```

PriceOscillator

Summary

The Price Oscillator calculates the spread between a short-period moving average and a long-period moving average.

Syntax

```
public interface PriceOscillator
```

Members

Name	Type	Summary
Result (25)	Property	The resulting time series of the PriceOscillator Indicator calculation

Example 1

```
//...
private PriceOscillator priceOscillator;

protected override void Initialize()
{
    priceOscillator = Indicators.PriceOscillator
        (Price, LongCycle, ShortCycle, MAType);
}
public override void Calculate(int index)
{
    double current = priceOscillator.Result[index];
}
```

Result

Summary

The resulting time series of the PriceOscillator Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private PriceOscillator priceOscillator;

protected override void Initialize()
{
    priceOscillator = Indicators.PriceOscillator
        (MarketSeries.Close, 14, 5, MovingAverageType.Simple);
}
public override void Calculate(int index)
{
    double result = priceOscillator.Result[index];
}
```

PriceROC

Summary

The Price ROC calculates the percentage change between the most recent price and the n-periods of past price.

Remarks

Can be used to determine whether an instrument is overbought or oversold.

Syntax

```
public interface PriceROC
```

Members

Name	Type	Summary
Result (26)	Property	The resulting time series of the PriceROC Indicator calculation

Example 1

```
private PriceROC _result;
protected override void Initialize()
{
    _result = Indicators.PriceROC(MarketSeries.Close, 14);
}
public override void Calculate(int index)
{
    double result = _result.Result[index];
}
```

Result

Summary

The resulting time series of the PriceROC Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private PriceROC _result;
protected override void Initialize()
{
    _result = Indicators.PriceROC(MarketSeries.Close, 14);
}
public override void Calculate(int index)
{
    double result = _result.Result[index];
}
```

PriceVolumeTrend

Summary

Price and Volume Trend is a variation of On Balance Volume, used to determine the strength of trends and warn of reversals.

Syntax

```
public interface PriceVolumeTrend
```

Members

Name	Type	Summary
Result (27)	Property	The time series of PriceVolumeTrend Indicator

Example 1

```
private PriceVolumeTrend _priceVolumeTrend;
[Parameter]
public DataSeries Source { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
   _priceVolumeTrend = Indicators.PriceVolumeTrend(Source);
}
public override void Calculate(int index)
{
   // Display Result of Indicator
   Result[index] = _priceVolumeTrend.Result[index];
}
```

Result

Summary

The time series of PriceVolumeTrend Indicator

Syntax

```
public IndicatorDataSeries Result{ get; set; }
```

Example 1

```
//...
private PriceVolumeTrend _priceVolumeTrend;
//...
protected override void OnBar()
{
    var currentValue = _priceVolumeTrend.Result.LastValue;
    //...
}
```

RainbowOscillator

Summary

Developed by Mel Widner, Rainbow Oscillator is based on multiple moving averages and helps to identify trends and provides overbought/oversold levels.

Syntax

```
public interface RainbowOscillator
```

Members

Name	Type	Summary
Result (28)	Property	The resulting time series of the RainbowOscillator Indicator calculation

Example 1

```
//...
private RainbowOscillator rainbow;
//...
protected override void Initialize()
{
    rainbow = Indicators.RainbowOscillator
        (MarketSeries.Close, 9, MovingAverageType.Simple);
    //...
}
public override void Calculate(int index)
{
    double result = rainbow.Result[index];
    //...
}
```

Result

Summary

The resulting time series of the RainbowOscillator Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
//...
```

```
private RainbowOscillator rainbow;
//...
protected override void Initialize()
{
    rainbow = Indicators.RainbowOscillator
        (MarketSeries.Close, 9, MovingAverageType.Simple);
    //...
}
public override void Calculate(int index)
{
    double result = rainbow.Result[index];
    //...
}
```

RelativeStrengthIndex

Summary

The RSI (Wilder) is momentum oscillator, measuring the velocity and magnitude of directional price movements.

Remarks

The RSI is most typically used on a 14 day timeframe, measured on a scale from 0 to 100, with high and low levels marked at 70 and 30, respectively. Shorter or longer timeframes are used for alternately shorter or longer outlooks. More extreme high and low levels—80 and 20, or 90 and 10—occur less frequently but indicate stronger momentum.

Syntax

```
public interface RelativeStrengthIndex
```

Members

Name	Type	Summary
Result (29)	Property	The resulting time series of the RelativeStrengthIndex Indicator calculation

Example 1

```
private RelativeStrengthIndex _rsi;
protected override void Initialize()
{
    _rsi = Indicators.RelativeStrengthIndex(MarketSeries.Close, 14);
}
public override void Calculate(int index)
{
```

```
    double result = _rsi.Result[index];
}
```

Result

Summary

The resulting time series of the RelativeStrengthIndex Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private RelativeStrengthIndex _rsi;
protected override void Initialize()
{
    _rsi = Indicators.RelativeStrengthIndex(MarketSeries.Close, 14);
}
public override void Calculate(int index)
{
    double result = _rsi.Result[index];
}
```

SimpleMovingAverage

Summary

The simple moving average is an average of price within n previous periods.

Remarks

The simple moving average is the unweighted mean of the previous n price data, where n is the period used for the calculation and price data the price data source, e.g. The closing price.

Syntax

```
public interface SimpleMovingAverage : MovingAverage, IIndicator
```

Members

Name	Type	Summary

Example 1

```
[Indicator]
public class SimpleMovingAverageExample : Indicator
{
    [Parameter]
    public DataSeries Source { get; set; }
    [Parameter(DefaultValue = 14, MinValue = 2)]
    public int Periods { get; set; }
    [Output("Result", Color = Colors.Orange)]
    public IndicatorDataSeries Result { get; set; }
    private SimpleMovingAverage _simpleMovingAverage;
    protected override void Initialize()
    {
        _simpleMovingAverage = Indicators.SimpleMovingAverage(Source, Periods);
    }
    public override void Calculate(int index)
    {
        var average = _simpleMovingAverage.Result[index];
        double sum = 0;
        for (var period = 0; period < Periods; period++)
        {
            sum += Math.Pow(Source[index - period] - average, 2.0);
        }
        Result[index] = Math.Sqrt(sum / Periods);
    }
}
```

StandardDeviation

Summary

Standard Deviation measures the market volatility with a commonly used statisctical function.

Syntax

```
public interface StandardDeviation
```

Members

Name	Type	Summary

Result (30)	Property	The resulting time series of the Standard Deviation Indicator calculation

Example 1

```
private StandardDeviation _standardDeviation;
protected override void Initialize()
{
    _standardDeviation = Indicators.StandardDeviation(MarketSeries.Close, 14,
MovingAverageType.Simple);
}
public override void Calculate(int index)
{
    double result = _standardDeviation.Result[index];
}
```

Result

Summary

The resulting time series of the Standard Deviation Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private StandardDeviation _standardDeviation;
protected override void Initialize()
{
    _standardDeviation = Indicators.StandardDeviation(MarketSeries.Close, 14,
MovingAverageType.Simple);
}
public override void Calculate(int index)
{
    double result = _standardDeviation.Result[index];
}
```

StochasticOscillator

Summary

The Stochastic Oscillator is a momentum indicator that aims to show price reversals by comparing the closing price to the price range.

Remarks

Calculates the range between the high and low price during a given period of time. The current price is then expressed as a percentage of this range with 0% indicating the bottom of the range and 100% indicating the top of the range over this time period. Based on the theory that prices tend to close near the boundaries of the recent range.

Syntax

```
public interface StochasticOscillator
```

Members

Name	Type	Summary
PercentD	Property	%D is 3 Period Exponential Moving Average of %K
PercentK	Property	Calculation of %K is 100 multiplied by the ratio of the closing price minus the lowest price over the last N periods over the highest price over the last N minus the lowest price over the last N periods.

Example 1

```
private StochasticOscillator _stochastic;
protected override void Initialize()
{
    // Initialize the Stochastic Oscillator indicator
     _stochastic = Indicators.StochasticOscillator(kPeriods, kSlowing, dPeriods, maType);
}
```

PercentD

Summary

%D is 3 Period Exponential Moving Average of %K

Syntax

```
public IndicatorDataSeries PercentD{ get; }
```

Example 1

```
double result = _stochastic.PercentK[index];
```

PercentK

Summary

Calculation of %K is 100 multiplied by the ratio of the closing price minus the lowest price over the last N periods over the highest price over the last N minus the lowest price over the last N periods.

Syntax

```
public IndicatorDataSeries PercentK{ get; }
```

Example 1

```
double result = _stochastic.PercentD[index];
```

SwingIndex

Summary

Developed by Welles Wilder, the Swing Index compares current Open, high, Low and Close prices to find of current and previous periods to find "real" price.

Syntax

```
public interface SwingIndex
```

Members

Name	Type	Summary
Result (31)	Property	The Result Series of the Swing Index Indicator

Example 1

```
using cAlgo.API;
```

```
using cAlgo.API.Indicators;
namespace cAlgo.Indicators
{
    [Indicator]
    public class Test:Indicator
    {
        private SwingIndex _swingIndex;
        [Parameter(DefaultValue = 12)]
        public int LimitMoveValue { get; set; }
        protected override void Initialize()
        {
            _swingIndex = Indicators.SwingIndex(LimitMoveValue);
        }

        public override void Calculate(int index)
        {
            //Print the current value of SwingIndex to the log
            Print("The current value of SwingIndex is {0}", _swingIndex.Result[index]);

        }
    }
}
```

Result

Summary

The Result Series of the Swing Index Indicator

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
public override void Calculate(int index)
{
     //Print the current value of SwingIndex to the log
     Print("The current value of SwingIndex is {0}", _swingIndex.Result[index]);
}
```

TimeSeriesMovingAverage

Summary

A Time Series Moving Average is moving average based on linear regression forecast.

Syntax

```
public interface TimeSeriesMovingAverage : MovingAverage, IIndicator
```

Members

Name	Type	Summary

Example 1

```
private TimeSeriesMovingAverage _timeSeriesMovingAverage;
protected override void Initialize()
{
    _timeSeriesMovingAverage = Indicators.TimeSeriesMovingAverage(MarketSeries.Close, 9);
}

public override void Calculate(int index)
{
    //Print the current value of TimeSeries Moving Average to the log
    Print("The current TimeSeries Moving Average  is {0}",
_timeSeriesMovingAverage.Result[index]);
}
```

TradeVolumeIndex

Summary

Trade Volume Index measures the amount of money flowing in and out of an asset.

Remarks

The underlying assumption of this indicator is that there is buying pressure when the price trades near the asking price and selling pressure when it trades near the bid.

Syntax

```
public interface TradeVolumeIndex
```

Members

Name	Type	Summary
Result (32)	Property	The time series of TradeVolumeIndex Indicator

Example 1

```
private TradeVolumeIndex _tradeVolume;
[Parameter]
public DataSeries Source { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
   _tradeVolume = Indicators.TradeVolumeIndex(Source);
}
public override void Calculate(int index)
{
   // Display Result of Indicator
   Result[index] = _tradeVolume.Result[index];
}
```

Result

Summary

The time series of TradeVolumeIndex Indicator

Syntax

```
public IndicatorDataSeries Result{ get; set; }
```

Example 1

```
//...
private TradeVolumeIndex _tradeVolume;
//...
protected override void OnBar()
{
    var currentValue = _tradeVolume.Result.LastValue;
    //...
}
```

TriangularMovingAverage

Summary

The Triangular Moving Average is a moving average that gives more weith to values located in the middle of aggregated period.

Syntax

```
public interface TriangularMovingAverage : MovingAverage, IIndicator
```

Members

Name	Type	Summary

Example 1

```
private TriangularMovingAverage _triangularMovingAverage;
protected override void Initialize()
{
    _triangularMovingAverage = Indicators.TriangularMovingAverage(MarketSeries.Close, 9);
}
public override void Calculate(int index)
{
    //Print the current value of _triangularMovingAverage to the log
    Print("The current Triangular Moving Average is {0}",
_triangularMovingAverage.Result[index]);
}
```

Trix

Summary

TRIX was developed by Jack Huton. It is a momentum oscillator that will help you filter unimportant price movement.

Remarks

When TRIX is rising, it is a good signal to buy, whether when TRIX is falling, it is a good signal to sell.

Syntax

```
```

```
public interface Trix
```

Members

Name	Type	Summary
Result (33)	Property	The resulting time series of the Trix Indicator calculation

Example 1

```
private Trix _result;
protected override void Initialize()
{
    _result = Indicators.Trix(MarketSeries.Close, 14);
}
public override void Calculate(int index)
{
    double result = _result.Result[index];
}
```

Result

Summary

The resulting time series of the Trix Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private Trix _result;
protected override void Initialize()
{
    _result = Indicators.Trix(MarketSeries.Close, 14);
}
public override void Calculate(int index)
{
    double result = _result.Result[index];
}
```

TrueRange

Summary

The Average True Range is a measure of market volatility developed by Wilder.

Syntax

```
public interface TrueRange
```

Members

Name	Type	Summary
Result (34)	Property	The resulting time series of the TrueRange Indicator calculation

Example 1

```
private TrueRange tri;

protected override void Initialize()
{
     tri = Indicators.TrueRange();
}
```

Result

Summary

The resulting time series of the TrueRange Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    Result[index] = tri.Result[index];
}
```

TypicalPrice

Summary

A Typical Price is an average of high, low and close values for a single period.

Remarks

Typical Price gives a simplified view of all prices for a period as a single series.

Syntax

```
public interface TypicalPrice
```

Members

Name	Type	Summary
Result (35)	Property	The resulting time series of the TypicalPrice Indicator calculation

Example 1

```
private TypicalPrice _result;
protected override void Initialize()
{
    _result = Indicators.TypicalPrice();
}
public override void Calculate(int index)
{
    double result = _result.Result[index];
}
```

Result

Summary

The resulting time series of the TypicalPrice Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private TypicalPrice _result;
protected override void Initialize()
{
    _result = Indicators.TypicalPrice();
}
public override void Calculate(int index)
{
    double result = _result.Result[index];
}
```

UltimateOscillator

Summary

The Ultimate Oscillator is a technical analysis oscillator based on a notion of buying or selling "pressure".

Remarks

It uses the weighted average of three different time periods to reduce the volatility and false transaction signals that are associated with many other indicators that mainly rely on a single time period.

Syntax

```
public interface UltimateOscillator
```

Members

Name	Type	Summary
Result (36)	Property	The resulting time series of the UltimateOscillator Indicator calculation

Example 1

```
private UltimateOscillator ultimateOscillator;
[Parameter("Cycle 1", DefaultValue = 7)]
public int Cycle1 { get; set; }
[Parameter("Cycle 2", DefaultValue = 14)]
public int Cycle2 { get; set; }
[Parameter("Cycle 3", DefaultValue = 28)]
```

```
public int Cycle3 { get; set; }
[Output("Main", Color = Colors.Green)]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
    ultimateOscillator = Indicators.UltimateOscillator(Cycle1,Cycle2,Cycle3);
}
public override void Calculate(int index)
{
    Result[index] = ultimateOscillator.Result[index];
}
```

Result

Summary

The resulting time series of the UltimateOscillator Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    double result = ultimateOscillator.Result[index];

    //...
}
```

VerticalHorizontalFilter

Summary

Vertical Horizontal Filter determines whether a price is going through a congestion phase or a trending phase.

Remarks

Vertical Horizontal Filter rises when trend is strong and falls when trend is weak.

Syntax

```
public interface VerticalHorizontalFilter
```

Members

Name	Type	Summary
Result (37)	Property	The resulting time series of the VerticalHorizontalFilter Indicator calculation

Example 1

```
//...
private VerticalHorizontalFilter VHFilter;
//...
protected override void Initialize()
{
    VHFilter = Indicators.VerticalHorizontalFilter(MarketSeries.Open, 28);
    //...
}
public override void Calculate(int index)
{
    double value = VHFilter.Result[index];
    //...
}
```

Result

Summary

The resulting time series of the VerticalHorizontalFilter Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    double result = VHFilter.Result[index];
    //...
}
```

Vidya

Summary

Volatility Index Dynamic Average (VIDYA) is a smoothing (moving average) based on dynamically changing periods.

Syntax

```
public interface Vidya : MovingAverage, IIndicator
```

Members

Name	Type	Summary

Example 1

```
[Parameter]
public DataSeries Price { get; set; }
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
[Parameter("Sigma", DefaultValue = 0.65, MinValue = 0.1, MaxValue = 0.95)]
public double Sigma { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
private Vidya vidya;
protected override void Initialize()
{
    vidya = Indicators.Vidya(Price, Period, Sigma);
}
public override void Calculate(int index)
{
    // Plot VIDYA to the chart
    Result[index] = vidya.Result.LastValue;
}
```

VolumeOscillator

Summary

The Volume Oscillator identifies trends in volume using a two moving average system. A strong trend is signaled when it

is positive. Falling volume indicates trend weakness.

Syntax

```
public interface VolumeOscillator
```

Members

Name	Type	Summary
Result (38)	Property	The time series of VolumeOscillator Indicator

Example 1

```
private VolumeOscillator _volumeOscillator;
[Parameter("Short Term", DefaultValue = 9)]
public int ShortTerm { get; set; }
[Parameter("Long Term", DefaultValue = 21)]
public int LongTerm { get; set; }
protected override void Initialize()
{
   _volumeOscillator = Indicators.VolumeOscillator(ShortTerm, LongTerm);
}
public override void Calculate(int index)
{
   // Display Result of Indicator
   Result[index] = _volumeOscillator.Result[index];
}
```

Result

Summary

The time series of VolumeOscillator Indicator

Syntax

```
public IndicatorDataSeries Result{ get; set; }
```

Example 1

```
//...
private VolumeOscillator _volumeOscillator;
```

```
//...
protected override void OnBar()
{
    var currentValue = _volumeOscillator.Result.LastValue;
    //...
}
```

VolumeROC

Summary

The Volume Rate of Change indicator measures the Rate Of Change of the tick volume.

Remarks

It shows whether or not a volume trend is developing and can be used to confirm price moves.

Syntax

```
public interface VolumeROC
```

Members

Name	Type	Summary
Result (39)	Property	The time series of VolumeROC Indicator

Example 1

```
private VolumeROC _volumeROC;
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
   _volumeROC = Indicators.VolumeROC(Period);
}
public override void Calculate(int index)
{
   // Display Result of Indicator
   Result[index] = _volumeROC.Result[index];
}
```

Result

Summary

The time series of VolumeROC Indicator

Syntax

```
public IndicatorDataSeries Result{ get; set; }
```

Example 1

```
//...
private VolumeROC _volumeROC;
//...
protected override void OnBar()
{
    var currentValue = _volumeROC.Result.LastValue;
    //...
}
```

WeightedClose

Summary

Weighted Close is an average of high, low and close prices where close has greater weight.

Remarks

Like a Typical price indicator weighted Close gives a simplified view of all prices for a period as a single series.

Syntax

```
public interface WeightedClose
```

Members

Name	Type	Summary
Result (40)	Property	The resulting time series of the WeightedClose Indicator calculation

Example 1

```
//...
private WeightedClose weightedCloseSeries;
//...
protected override void Initialize()
{
    weightedCloseSeries = Indicators.WeightedClose();
    //...
}
public override void Calculate(int index)
{
    double weightedCloseValue = weightedCloseSeries.Result[index];
    //...
}
```

Result

Summary

The resulting time series of the WeightedClose Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
double weightedCloseValue = weightedCloseSeries.Result[index];
```

WeightedMovingAverage

Summary

The Weighted Moving Average is a moving average that gives more weith to the latest values.

Syntax

```
public interface WeightedMovingAverage : MovingAverage, IIndicator
```

Members

Name	Type	Summary
Result (41)	Property	The resulting time series of the WeightedMovingAverage Indicator calculation

Example 1

```
private WeightedMovingAverage _weightedMovingAverage;
protected override void OnStart()
{
    _weightedMovingAverage = Indicators.WeightedMovingAverage(Source, Period);
}
protected override void OnTick()
{
    if(Trade.IsExecuting)
        return;
    int index = MarketSeries.Close.Count - 1;
    if(Symbol.Bid > _weightedMovingAverage.Result[index])
    {
        Trade.CreateMarketOrder(TradeType.Buy, Symbol, Volume);
    }
 }
```

Result

Summary

The resulting time series of the WeightedMovingAverage Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private WeightedMovingAverage _weightedMovingAverage;
protected override void OnStart()
{
    _weightedMovingAverage = Indicators.WeightedMovingAverage(Source, Period);
}
protected override void OnTick()
{
    if(Trade.IsExecuting)
```

```
        return;
    int index = MarketSeries.Close.Count - 1;
    if(Symbol.Bid > _weightedMovingAverage.Result[index])
    {
        Trade.CreateMarketOrder(TradeType.Buy, Symbol, Volume);
    }
 }
```

WellesWilderSmoothing

Summary

The Welles Wilder's Smoothing indicator is an exponential moving average, but it has different alpha ration. As a result it responds to price changes slower.

Remarks

Usage is the same as EMA usage. Please mind the different in alpha ration.

Syntax

```
public interface WellesWilderSmoothing : MovingAverage, IIndicator
```

Members

Name	Type	Summary

Example 1

```
private WellesWilderSmoothing _result;
protected override void Initialize()
{
    _result = Indicators.WellesWilderSmoothing(MarketSeries.Close, 14);
}
public override void Calculate(int index)
{
    double result = _result.Result[index];
}
```

WilliamsAccumulationDistribution

Summary

William's Accumulation Distribution is an oscillator that can identify if the market is driven by buyers (accumulation) or by sellers (distribution)

Remarks

The divergence between price and the William's Accumulation Distribution. When price is falling and WAD is rising, it is a buying opportunity

Syntax

```
public interface WilliamsAccumulationDistribution
```

Members

Name	Type	Summary
Result (42)	Property	The resulting time series of the WilliamsAccumulationDistribution Indicator calculation

Example 1

```
private WilliamsAccumulationDistribution _result;
protected override void Initialize()
{
    _result = Indicators.WilliamsAccumulationDistribution();
}
public override void Calculate(int index)
{
    double result = _result.Result[index];
}
```

Result

Summary

The resulting time series of the WilliamsAccumulationDistribution Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private WilliamsAccumulationDistribution _result;
protected override void Initialize()
{
    _result = Indicators.WilliamsAccumulationDistribution();
}
public override void Calculate(int index)
{
    double result = _result.Result[index];
}
```

WilliamsPctR

Summary

Williams %R is an effective momentum oscillator and was described by Larry Williams for the first time in 1973.

Remarks

It shows the relationship of the close relative to the high-low range over a set period of time.

Syntax

```
public interface WilliamsPctR
```

Members

Name	Type	Summary
Result (43)	Property	The resulting time series of the WilliamsPctR Indicator calculation

Example 1

```
private WilliamsPctR _result;
protected override void Initialize()
{
    _result = Indicators.WilliamsPctR(14);
}
public override void Calculate(int index)
{
    double result = _result.Result[index];
}
```

Result

Summary

The resulting time series of the WilliamsPctR Indicator calculation

Syntax

```
public IndicatorDataSeries Result{ get; }
```

Example 1

```
private WilliamsPctR _result;
protected override void Initialize()
{
    _result = Indicators.WilliamsPctR(14);
}
public override void Calculate(int index)
{
    double result = _result.Result[index];
}
```

All classes in cAlgo.API.Internals

Name	Type	Summary
AccountType	Enum	Returns current account type
Algo	Class	The container class for the main cAlgo.API Interfaces
IAccount	Interface	Contains the current account information.
IIndicator	Interface	Base interface for all Indicators
IIndicatorsAccessor	Interface	Accessor to Indicators
INotifications	Interface	It is an interface that represents all Notifications.
IServer	Interface	Server related information.
ISmallScriptsController	Interface	-
LeverageTier	Interface	Tier of dynamic leverage.
MarketData	Interface	Provides access to Depth of Market Data
MarketHours	Interface	Access to symbol's trading sessions schedule
		Provides access to the market data such as the DataSeries Open, High, Low,

MarketSeries	Interface	Close.
Symbol	Interface	Represents a currency pair
TradingSession	Interface	Trading session schedule

AccountType

Summary

Returns current account type

Syntax

```
public sealed enum AccountType
```

Members

Name	Type	Summary
Hedged	Field	Account type that allows hedged positions
Netted	Field	Account type that allows only single net position per symbol

Hedged

Summary

Account type that allows hedged positions

Syntax

```
AccountType.Hedged
```

Netted

Summary

Account type that allows only single net position per symbol

Syntax

```
AccountType.Netted
```

Algo

Summary

The container class for the main cAlgo.API Interfaces

Syntax

```
public class Algo : Object
```

Members

Name	Type	Summary
Algo	Method	
BeginInvokeOnMainThread	Method	Invokes asynchronously the specified code on the main cBot or Indicator thread.
Chart (10)	Property	Represents the chart where cBot or Indicator is launched.
CreateDataSeries	Method	Initialization of an IndicatorDataSeries.
History (2)	Property	Represents the collection of all historical trades of the account.
Indicators	Property	Access to built-in Indicators.
IsBacktesting	Property	True if the robot is in Backtesting mode, false otherwise
MarketData	Property	Provides access to Depth of Market Data
MarketSeries (2)	Property	Market series of the current symbol and time frame.
Notifications	Property	Represents notifications such as sounds and email
OnTimer	Method	Called when the timer interval has elapsed.
PendingOrders (2)	Property	Array of all Pending Orders of the account
Positions (3)	Property	Collection of all open positions of the account
Print	Method	Prints a message to the Log
RefreshData	Method	Updates MarketSeries, Positions, PendingOrders, History, etc.
RunningMode	Property	
Server	Property	Server related information.
Symbol (2)	Property	Represents the current symbol provides access to its properties and certain methods

Time (4)	Property	Returns the current server time. The shortcut to the Server.Time property.
TimeFrame (2)	Property	Access to TimeFrame values
Timer	Property	Access to the Timer object.
TimeZone (2)	Property	TimeZone of cBot or Indicator

MarketSeries

Summary

Market series of the current symbol and time frame.

Remarks

Access to Open, High, Low, Close, Typical, Median and Weighted Price, Open Time and current Time frame.

Syntax

```
public MarketSeries MarketSeries{ get; }
```

Example 1

```
//Access price and time data through MarketSeries
var closePrice = MarketSeries.Close[index];
var openTime = MarketSeries.OpenTime.LastValue;
```

Indicators

Summary

Access to built-in Indicators.

Syntax

```
public IIndicatorsAccessor Indicators{ get; }
```

Example 1

```
protected override void Initialize()
```

```
{
    //Use MarketSeries price data as parameters to indicators
    _ma = Indicators.SimpleMovingAverage(MarketSeries.Close, 20);
}
```

Symbol

Summary

Represents the current symbol provides access to its properties and certain methods

Syntax

```
public Symbol Symbol{ get; }
```

Example 1

```
var ask = Symbol.Ask;
var bid = Symbol.Bid;
var digits = Symbol.Digits;
var pip = Symbol.PipSize;
var maxVolume = Symbol.VolumeMax;
var minVolume = Symbol.VolumeMin;
```

Example 2

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10, 10);
```

Example 3

```
volume = Symbol.NormalizeVolume(volume, RoundingMode.Down);
```

Notifications

Summary

Represents notifications such as sounds and email

Syntax

```
public INotifications Notifications{ get; }
```

Example 1

```
Notifications.PlaySound(@"C:\Windows\Media\notify.wav");
```

Example 2

```
string emailBody = "this is the message send";
Notifications.SendEmail("from@example.com", "to@example.com", "my subject", emailBody);
```

TimeFrame

Summary

Access to TimeFrame values

Syntax

```
public TimeFrame TimeFrame{ get; }
```

Example 1

```
if(TimeFrame == TimeFrame.Daily)
{
   //...
}
```

Server

Summary

Server related information.

Syntax

```
public IServer Server{ get; }
```

Example 1

```
protected override void OnTick()
{
    Print("The server time is: {0}", Server.Time);
}
```

MarketData

Summary

Provides access to Depth of Market Data

Syntax

```
public MarketData MarketData{ get; }
```

Example 1

```
private MarketDepth _md;
_md = MarketData.GetMarketDepth("GBPUSD");
```

IsBacktesting

Summary

True if the robot is in Backtesting mode, false otherwise

Syntax

```
public bool IsBacktesting{ get; }
```

Example 1

```
if(IsBacktesting)
{
   Print(MarketSeries.OpenTime.LastValue);
}
```

TimeZone

Summary

TimeZone of cBot or Indicator

Syntax

```
public TimeZoneInfo TimeZone{ get; }
```

Positions

Summary

Collection of all open positions of the account

Syntax

```
public Positions Positions{ get; }
```

Example 1

```
foreach (var position in Positions)
{
    if (position.StopLoss == null)
        ModifyPosition(position, 10, position.TakeProfit);
}
```

PendingOrders

Summary

Array of all Pending Orders of the account

Syntax

```
public PendingOrders PendingOrders{ get; }
```

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfit,
                           order.ExpirationTime);
}
```

History

Summary

Represents the collection of all historical trades of the account.

Syntax

```
public History History{ get; }
```

Example 1

```
foreach (HistoricalTrade trade in History)
{
    Print(trade.EntryTime);
}
```

Timer

Summary

Access to the Timer object.

Syntax

```
public Timer Timer{ get; }
```

Example 1

```
protected override void OnStart()
{
    Timer.Start(1);//start timer with 1 second interval
}
protected override void OnTimer()
{
    ChartObjects.DrawText("time", Time.ToString("HH:mm:ss"), StaticPosition.TopLeft);
}
```

Time

Summary

Returns the current server time. The shortcut to the Server.Time property.

Syntax

```
public DateTime Time{ get; }
```

Example 1

```
protected override void OnTick()
{
    Print("The Server Time is: {0}", Time);
}
```

Chart

Summary

Represents the chart where cBot or Indicator is launched.

Syntax

```
public Chart Chart{ get; }
```

RunningMode

Syntax

```
public RunningMode RunningMode{ get; }
```

Print

Summary

Prints a message to the Log

Syntax

```
public void Print(string message, Object[] parameters)
```

```
public void Print(Object[] parameters)
```

```
public void Print(Object value)
```

Parameters

Name	Description

Example 1

```
Print("Current Balance is {0}, Equity is {1}.", Account.Balance, Account.Equity);
```

Print

Summary

Prints a message to the Log

Syntax

```
public void Print(string message, Object[] parameters)
```

```
public void Print(Object[] parameters)
```

```
public void Print(Object value)
```

Parameters

Name	Description

Example 1

```
Print(Account.Balance, " ", Account.Equity);
```

Print

Summary

Prints text representation of the specified object to log.

Syntax

```
public void Print(string message, Object[] parameters)
```

```
public void Print(Object[] parameters)
```

```
public void Print(Object value)
```

Parameters

Name	Description

Example 1

```
Print(Account.Positions.Count);
```

CreateDataSeries

Summary

Initialization of an IndicatorDataSeries.

Syntax

```
public IndicatorDataSeries CreateDataSeries()
```

Example 1

```
private IndicatorDataSeries series;
protected override void Initialize()
{
    series = CreateDataSeries();
}
public override void Calculate(int index)
{
    series[index] = (MarketSeries.Close[index] + MarketSeries.Open[index]) / 2;
}
```

RefreshData

Summary

Updates MarketSeries, Positions, PendingOrders, History, etc.

Syntax

```
public void RefreshData()
```

OnTimer

Summary

Called when the timer interval has elapsed.

Syntax

```
protected virtual void OnTimer()
```

Example 1

```
protected override void OnStart()
{
    Timer.Start(1);//start timer with 1 second interval
}
protected override void OnTimer()
{
    ChartObjects.DrawText("time", Time.ToString("HH:mm:ss"), StaticPosition.TopLeft);
}
```

BeginInvokeOnMainThread

Summary

Invokes asynchronously the specified code on the main cBot or Indicator thread.

Syntax

```
public void BeginInvokeOnMainThread(Action action)
```

Parameters

Name	Description

Algo

Syntax

```
public Algo Algo()
```

IAccount

Summary

Contains the current account information.

Syntax

```
public interface IAccount
```

Members

Name	Type	Summary
AccountType	Property	Returns the current account type.
Balance (2)	Property	Returns the balance of the current account.
BrokerName	Property	Returns the broker name of the current account.
Currency	Property	Returns the currency of the current account, e.g. "EUR".
Equity (2)	Property	Represents the equity of the current account (balance minus Unrealized Net Loss plus Unrealized Net Profit plus Bonus).
FreeMargin	Property	Represents the free margin of the current account.
IsLive	Property	Defines if the account is Live or Demo. True if the Account is Live, False if it is a Demo.
Margin	Property	Represents the margin of the current account.
MarginLevel	Property	Represents the margin level of the current account. Margin level (in %) is calculated using this formula: Equity / Margin * 100
Number	Property	Returns the number of the current account, e.g. 123456.
PreciseLeverage	Property	Gets the precise account leverage value.
StopOutLevel	Property	Stop Out level is a lowest allowed Margin Level for account. If Margin Level is less than Stop Out, position will be closed sequentially until Margin Level is greater than

		Stop Out.
UnrealizedGrossProfit	Property	Gets the Unrealized Gross profit value.
UnrealizedNetProfit	Property	Gets the Unrealized Gross profit value.

Example 1

```
// Account Properties
// Current Account Balance
double balance = Account.Balance;
// Current Account Currency e.g. EUR
string currency = Account.Currency;
// Current Account Equity
double equity = Account.Equity;
// Current Account Free Margin
double freemargin = Account.FreeMargin;
// Current Account Margin
double margin = Account.Margin;
//Margin level = Equity / Margin * 100
double? marginlevel = Account.MarginLevel;
```

AccountType

Summary

Returns the current account type.

Syntax

```
public AccountType AccountType{ get; }
```

Balance

Summary

Returns the balance of the current account.

Syntax

```
public double Balance{ get; }
```

Example 1

```
double balancebefore;
double balanceafter;
protected override void OnStart()
{
   // store the balance upon start up of the robot
   balancebefore = Account.Balance;
}
protected override void OnStop()
{
    // Store the balance upon stop of the robot.
    balanceafter = Account.Balance;
    // print the difference
        Print("The difference of balancebefore and balanceafter is: {0}",
balancebefore-balanceafter);
}
```

Example 2

```
if ( Account.Balance < 0 )
    Stop();
```

Currency

Summary

Returns the currency of the current account, e.g. "EUR".

Syntax

```
public string Currency{ get; }
```

Example 1

```
Print("The currency of the current account is: {0}", Account.Currency);
```

Equity

Summary

Represents the equity of the current account (balance minus Unrealized Net Loss plus Unrealized Net Profit plus Bonus).

Syntax

```
public double Equity{ get; }
```

Example 1

```
Print("The equity of this account is: {0}", Account.Equity);
```

Margin

Summary

Represents the margin of the current account.

Syntax

```
public double Margin{ get; }
```

Example 1

```
Print("The margin of this account is: {0}", Account.Margin);
```

FreeMargin

Summary

Represents the free margin of the current account.

Syntax

```
public double FreeMargin{ get; }
```

Example 1

```
Print("The free margin of this account is: {0}", Account.FreeMargin);
```

MarginLevel

Summary

Represents the margin level of the current account. Margin level (in %) is calculated using this formula: Equity / Margin * 100

Syntax

```
public double? MarginLevel{ get; }
```

Example 1

```
Print("The marginlevel of this account is: {0}", Account.MarginLevel);
```

IsLive

Summary

Defines if the account is Live or Demo. True if the Account is Live, False if it is a Demo.

Syntax

```
public bool IsLive{ get; }
```

Example 1

```
if (Account.IsLive)
    Print("Live Account");
else
    Print("Demo Account");
```

Number

Summary

Returns the number of the current account, e.g. 123456.

Syntax

```
public int Number{ get; }
```

BrokerName

Summary

Returns the broker name of the current account.

Syntax

```
public string BrokerName{ get; }
```

UnrealizedGrossProfit

Summary

Gets the Unrealized Gross profit value.

Syntax

```
public double UnrealizedGrossProfit{ get; }
```

UnrealizedNetProfit

Summary

Gets the Unrealized Gross profit value.

Syntax

```
public double UnrealizedNetProfit{ get; }
```

PreciseLeverage

Summary

Gets the precise account leverage value.

Syntax

```
public double PreciseLeverage{ get; }
```

Example 1

```
var leverage = Account.Leverage;
```

StopOutLevel

Summary

Stop Out level is a lowest allowed Margin Level for account. If Margin Level is less than Stop Out, position will be closed sequentially until Margin Level is greater than Stop Out.

Syntax

```
public double StopOutLevel{ get; }
```

IIndicator

Summary

Base interface for all Indicators

Syntax

```
public interface IIndicator
```

Members

Name	Type	Summary
Calculate (2)	Method	Method to calculate the value(s) of indicator for given index.

Calculate

Summary

Method to calculate the value(s) of indicator for given index.

Syntax

```
public void Calculate(int index)
```

Parameters

Name	Description

Example 1

```
Example 1
//...
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
//...
public override void Calculate(int index)
{
    // Calculate value at specified index

    // if the index is less than Period exit
    if(index < Period)
        return;
    // Maximum returns the largest number in the Series in the range [Series[index-Period],
Series[index]]
    double high = MarketSeries.High.Maximum(Period);
    // Minimum returns the smallest number in the Series in the range [index - Period, index]
    double low = MarketSeries.Low.Minimum(Period);
```

```
    double center = (high + low) / 2;
    // Display Result of Indicator
    Result[index] = center;
}
Example 2
//...
[Parameter]
public DataSeries Source { get; set; }
[Parameter("Periods", DefaultValue = 25)]
public int Periods { get; set; }
//...
public override void Calculate(int index)
{
    // Simple moving average calculation
    double sum = 0.0;
    for (int i = index - Periods + 1; i <= index; i++)
    {
        sum += Source[i];
    }
    Result[index] = sum / Periods;

}
//...
```

IIndicatorsAccessor

Summary

Accessor to Indicators

Syntax

```
public interface IIndicatorsAccessor
```

Members

Name	Type	Summary
AcceleratorOscillator	Method	Initializes the AcceleratorOscillator indicator instance
AccumulativeSwingIndex	Method	Initializes the Accumulative Swing Index indicator
Aroon	Method	The Aroon indicator is used to identify trends and their reversals.
AverageTrueRange	Method	Average true range. An indicator providing the degree of price volatility.
AwesomeOscillator	Method	Initializes the AwesomeOscillator indicator instance
BollingerBands	Method	The Bollinger Bands indicator shows volatility.

ChaikinMoneyFlow	Method	The Chaikin Money Flow indicator measures the money flow volume over a specific period.
ChaikinVolatility	Method	The Chaikin Volatiliy indicator measures the trading range between the high and the low prices.
CommodityChannelIndex	Method	The Commodity Channel Index identifies overbough and oversold conditions, price reversals and trend strength.
DetrendedPriceOscillator	Method	The Detrended Price Oscillator shows intermediate overbought and oversold levels.
DirectionalMovementSystem	Method	The Directional Movement System is composed of three indicators that show if the market is trending and provide signals.
DonchianChannel	Method	The Donchian channel is a volatility indicator forming a channel between the highest high and the lowest low of the chosen period.
EaseOfMovement	Method	The Ease Of Movement indicator relates the price change to the volume.
ExponentialMovingAverage	Method	The Exponential Moving Average smoothes the price data producing a trend indicator.
FractalChaosBands	Method	The Fractal Chaos Bands indicator breaks down large trends into predictable patterns.
GetIndicator	Method	Initializes the custom indicator
HighMinusLow	Method	The High Minus Low indicator is used to compute the range of daily bars
HistoricalVolatility	Method	The Historical Volatility indicator is derived from time series of past market prices.
IchimokuKinkoHyo	Method	Ichimoku Kinko Hyo Indicator is a moving average based trend identification system.
KeltnerChannels	Method	Initializes the Keltner Channels indicator instance
LinearRegressionForecast	Method	Linear Regression Forecast is a trend indicator used to forecast values using the Least Squares Fit method.
LinearRegressionIntercept	Method	The Linear Regression Intercept can be used together with the Linear Regression Slope indicator to plot the Linear Regression Line.
LinearRegressionRSquared	Method	The R Squared or coefficient of determination indicator's main purpose is the confirm the strength of the market.
LinearRegressionSlope	Method	The Linear Regression Slope indicator is intended to measure the direction and strength of a trend.
MacdCrossOver	Method	The MACD Line with the Signal line and their difference as a histogram.
MacdHistogram	Method	The MACD Histogram is a momentum indicator measured by typically subtracting a 26 period moving average from a 12 period moving average.
MassIndex	Method	The Mass Index indicator is used to predict trend reversals.
MedianPrice	Method	The Median indicator is the average of the high and the low.
MomentumOscillator	Method	The Momentum Oscillator measures the momentum of the price.
MoneyFlowIndex	Method	The Money Flow Index measures the strength of the money flow.

MovingAverage	Method	Moving Average indicators are used to smooth data producing trend indicators.
NegativeVolumeIndex	Method	The Negative Volume Index is a calculation of the percentage change in price on days when trading volume declines.
OnBalanceVolume	Method	The On Balance Volume indicator relates price and volume.
ParabolicSAR	Method	The Parabolic SAR indicator identifies potential reversals in the market direction
PositiveVolumeIndex	Method	The Positive Volume Index is a calculation of the percentage change in price on days when trading volume increased.
PriceOscillator	Method	The Price Oscillator calculates the difference between two moving averages.
PriceROC	Method	The Price Rate of Change indicator is the percentage change of the current price and the price N periods ago.
PriceVolumeTrend	Method	The Price Volume Trend indicator shows the relationship between price and volume.
RainbowOscillator	Method	The Rainbow Oscillator is a process of repetitive smoothing of simple moving averages resulting in a full spectrum of trends.
RelativeStrengthIndex	Method	The Relative Strength Index indicator measures turns in price by measuring turns in momentum.
SimpleMovingAverage	Method	The simple moving average smoothes the price data producing a trend indicator
StandardDeviation	Method	The Standard Deviation indicator shows volatility.
StochasticOscillator	Method	The Stochastic Oscillator is a momentum indicator that aims to show price reversals by comparing the closing price to the price range.
SwingIndex	Method	Returns the Swing Index indicator instance.
TimeSeriesMovingAverage	Method	The Time Series Moving Average is a moving average based on linear regression
TradeVolumeIndex	Method	Trade Volume Index indicator measures the amount of money flowing in and out of an asset.
TriangularMovingAverage	Method	The Triangular Moving Average is averaged twice to produce a double smoothed trend indicator
Trix	Method	The Trix indicator shows the slope of a triple-smoothed exponential moving average.
TrueRange	Method	Initializes the True Range indicator.
TypicalPrice	Method	The Typical Price indicator is the average of the high, low, and closing prices.
UltimateOscillator	Method	Returns the Ultimate Oscillator indicator instance.
VerticalHorizontalFilter	Method	The Vertical Horizontal Filter indicator measures the level of trend activity.
Vidya	Method	Volatility Index Dynamic Average (VIDYA) is a smoothing (moving average) based on dynamically changing periods.
VolumeOscillator	Method	The Volume Oscillator indicator is the difference between two moving averages.

VolumeROC	Method	Volume Rate of Change Indicator measures the rate of change of the tick volume.
WeightedClose	Method	The WeightedClose indicator is an average of each day's price with extra weight given to the closing price.
WeightedMovingAverage	Method	The Weighted Moving Average smoothes the price data producing a trend indicator.
WellesWilderSmoothing	Method	Welles Wilder Smoothing eliminates noise to identify the trend.
WilliamsAccumulationDistribution	Method	The Williams Accumulation Distribution indicator shows bullish or bearish trends.
WilliamsPctR	Method	The Williams Percent R indicator is a momentum indicator measuring overbought and oversold levels.

GetIndicator

Summary

Initializes the custom indicator

Syntax

```
public TIndicator GetIndicator(Object[] parameterValues)
```

```
public TIndicator GetIndicator(MarketSeries marketSeries, Object[] parameterValues)
```

Parameters

Name	Description

Example 1

```
private SampleSMA sma;
protected override void Initialize()
{
    sma = Indicators.GetIndicator<SampleSMA>(Source, Period);
}
public override void Calculate(int index)
{
    // Display the sma result on the chart
    Result[index] = sma.Result[index];
}
```

GetIndicator

Summary

Initializes the custom indicator for a specific timeframe

Syntax

```
public TIndicator GetIndicator(Object[] parameterValues)
```

```
public TIndicator GetIndicator(MarketSeries marketSeries, Object[] parameterValues)
```

Parameters

Name	Description

Example 1

```
private AdaptiveCG adaptiveCG;
private MarketSeries seriesMin5;
protected override void Initialize()
{
    seriesMin5 = MarketData.GetSeries(TimeFrame.Minute5);
    adaptiveCG = Indicators.GetIndicator<AdaptiveCG>(seriesMin5, Alpha);
}
```

MovingAverage

Summary

Moving Average indicators are used to smooth data producing trend indicators.

Syntax

```
public MovingAverage MovingAverage(DataSeries source, int periods, MovingAverageType maType)
```

Parameters

Name	Description

Example 1

```
private MovingAverage ma;
protected override void Initialize()
{
    ma = Indicators.MovingAverage(MarketSeries.Close, 50, MovingAverageType.Simple);
}
public override void Calculate(int index)
{
    // Display the ma result on the chart
    Result[index] = ma.Result[index];
}
```

ExponentialMovingAverage

Summary

The Exponential Moving Average smoothes the price data producing a trend indicator.

Syntax

```
public ExponentialMovingAverage ExponentialMovingAverage(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
private ExponentialMovingAverage ema;
protected override void Initialize()
{
    ema = Indicators.ExponentialMovingAverage(MarketSeries.Close, 50);
}
public override void Calculate(int index)
{
    // Display the ema result on the chart
    Result[index] = ema.Result[index];
}
```

WeightedMovingAverage

Summary

The Weighted Moving Average smoothes the price data producing a trend indicator.

Syntax

```
public WeightedMovingAverage WeightedMovingAverage(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
private WeigthedMovingAverage wma;
protected override void Initialize()
{
    wma = Indicators.WeightedMovingAverage(MarketSeries.Close, 20);
}
public override void Calculate(int index)
{
    Result[index] = wma.Result[index];
}
```

SimpleMovingAverage

Summary

The simple moving average smoothes the price data producing a trend indicator

Syntax

```
public SimpleMovingAverage SimpleMovingAverage(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
private SimpleMovingAverage sma;
protected override void Initialize()
{
    sma = Indicators.SimpleMovingAverage(MarketSeries.Close, 14);
}
public override void Calculate(int index)
{
    Result[index] = sma.Result[index];
}
```

TriangularMovingAverage

Summary

The Triangular Moving Average is averaged twice to produce a double smoothed trend indicator

Syntax

```
public TriangularMovingAverage TriangularMovingAverage(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
[Parameter]
public DataSeries Source { get; set; }
[Output("Result", Color = Colors.Orange)]
public IndicatorDataSeries Result { get; set; }
private SimpleMovingAverage tma;
protected override void Initialize()
{
    tma = Indicators.TriangularMovingAverage(Source, 10);
}
public override void Calculate(int index)
{
    Result[index] = tma.Result[index];
}
```

HighMinusLow

Summary

The High Minus Low indicator is used to compute the range of daily bars

Syntax

```
public HighMinusLow HighMinusLow()
```

```
public HighMinusLow HighMinusLow(MarketSeries marketSeries)
```

Example 1

```
[Output("Main")]
public IndicatorDataSeries Result {get; set;}
private HighMinusLow highMinusLow;
protected override void Initialize()
{
    highMinusLow = Indicators.HighMinusLow();
}
public override void Calculate(int index)
{
    // Display the High Minus Low indicator on the chart
    Result[index] = highMinusLow.Result[index];
    Print("Previous HighMinusLow is: {0}", highMinusLow.Result[index-1]);
}
//...
```

HighMinusLow

Summary

Initializes the High Minus Low indicator for a specific timeframe

Remarks

The High Minus Low indicator is used to compute the range of daily bars

Syntax

```
public HighMinusLow HighMinusLow()
```

```
public HighMinusLow HighMinusLow(MarketSeries marketSeries)
```

Parameters

Name	Description

Example 1

```
private MarketSeries seriesMin5;
private HighMinusLow highMinusLow;
protected override void Initialize()
{
    seriesMin5 = MarketData.GetSeries(TimeFrame.Minute5);
    highMinusLow = Indicators.HighMinusLow(seriesMin5);
}
public override void Calculate(int index)
{
    //...
    Print("Min 5 HighMinusLow is: {0}", highMinusLow.Result[indexMin5]);
}
```

TrueRange

Summary

Initializes the True Range indicator.

Remarks

The True Range indicator is the daily range plus any gap from the closing price of the previous day

Syntax

```
public TrueRange TrueRange()
```

```
public TrueRange TrueRange(MarketSeries marketSeries)
```

Example 1

```

```

```
[Output("Main")]
public IndicatorDataSeries Result {get; set;}
private TrueRange trueRange;
protected override void Initialize()
{
    trueRange = Indicators.TrueRange();
}
public override void Calculate(int index)
{
    Result[index] = trueRange.Result[index];
}
```

TrueRange

Summary

Initializes the True Range indicator for a specific timeframe

Remarks

The True Range indicator is the daily range plus any gap from the closing price of the previous day

Syntax

```
public TrueRange TrueRange()
```

```
public TrueRange TrueRange(MarketSeries marketSeries)
```

Parameters

Name	Description

Example 1

```
protected override void Initialize()
{
    seriesMin10 = MarketData.GetSeries(TimeFrame.Minute10);
    trueRange = Indicators.TrueRange(seriesMin10);
}
```

WellesWilderSmoothing

Summary

Welles Wilder Smoothing eliminates noise to identify the trend.

Syntax

```
public WellesWilderSmoothing WellesWilderSmoothing(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
[Output("Main")]
public IndicatorDataSeries Result {get; set;}
private WellesWilderSmoothing wws;
protected override void Initialize()
{
    wws = Indicators.WellesWilderSmoothing(MarketSeries.Close, 14);
}
public override void Calculate(int index)
{
    Result[index] = wws.Result[index];
}
```

SwingIndex

Summary

Returns the Swing Index indicator instance.

Syntax

```
public SwingIndex SwingIndex(int limitMoveValue)
```

```
public SwingIndex SwingIndex(MarketSeries marketSeries, int limitMoveValue)
```

Parameters

Name	Description

Example 1

```
[Parameter(DefaultValue = 20)]
public int limitMove { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
private SwingIndex si;
protected override void Initialize()
{
   si = Indicators.SwingIndex(limitMove);
}
public override void Calculate(int index)
{
    //This stores current SwingIndex to Result Output
    Result[index] = si.Result[index];
    // This prints previous SwingIndex to log
    Print("Previous SwingIndex is: {0}", si.Result[index-1]);
}
```

SwingIndex

Summary

Returns the Swing Index indicator instance.

Syntax

```
public SwingIndex SwingIndex(int limitMoveValue)
```

```
public SwingIndex SwingIndex(MarketSeries marketSeries, int limitMoveValue)
```

Parameters

Name	Description

Example 1

```
protected override void Initialize()
```

```
{
   seriesMin10 = MarketData.GetSeries(TimeFrame.Minute10);
   swingIndex = Indicators.SwingIndex(seriesMin10, limitMove);
}
```

AccumulativeSwingIndex

Summary

Initializes the Accumulative Swing Index indicator

Remarks

The Accumulative Swing Index indicator is used as a divergence and confirmation tool.

Syntax

```
public AccumulativeSwingIndex AccumulativeSwingIndex(int limitMoveValue)
```

```
public AccumulativeSwingIndex AccumulativeSwingIndex(MarketSeries marketSeries, int
limitMoveValue)
```

Parameters

Name	Description

Example 1

```
[Parameter(DefaultValue = 20)]
public int limitMove { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
private AccumulativeSwingIndex asi;
protected override void Initialize()
{
   asi = Indicators.AccumulativeSwingIndex(limitMove);
}
public override void Calculate(int index)
{
    //This stores current AccumulativeSwingIndex to Result Output
    Result[index] = asi.Result[index];
    // This prints previous AccumulativeSwingIndex to log
    Print("Previous AccumulativeSwingIndex is: {0}", asi.Result[index-1]);
}
```

AccumulativeSwingIndex

Summary

Initializes the Accumulative Swing Index indicator for a specific timeframe

Syntax

```
public AccumulativeSwingIndex AccumulativeSwingIndex(int limitMoveValue)
```

```
public AccumulativeSwingIndex AccumulativeSwingIndex(MarketSeries marketSeries, int
limitMoveValue)
```

Parameters

Name	Description

Example 1

```
protected override void Initialize()
{
   dailySeries = MarketData.GetSeries(TimeFrame.Daily);
   accumulativeSwingIndex = Indicators.AccumulativeSwingIndex(dailySeries, limitMove);
}
```

Aroon

Summary

The Aroon indicator is used to identify trends and their reversals.

Syntax

```
public Aroon Aroon(int periods)
```

```
public Aroon Aroon(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

Example 1

```
[Parameter("Period")]
public int Period { get; set; }
private Aroon aroon;
protected override void OnStart()
{
    aroon = Indicators.Aroon(Period);
}
protected override void OnTick()
{
    if (aroon.Up.LastValue < aroon.Down.LastValue)
    {
        //Do something
    }
}
```

Aroon

Summary

Initializes the Aroon indicator instance

Syntax

```
public Aroon Aroon(int periods)
```

```
public Aroon Aroon(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

Example 1

```
[Parameter("Period")]
```

```
public int Period { get; set; }
private Aroon aroonDaily;
private MarketSeries dailySeries;
protected override void OnStart()
{
    dailySeries = MarketData.GetSeries(TimeFrame.Daily);
    aroonDaily = Indicators.Aroon(dailySeries, Period);
}
protected override void OnTick()
{
    if (aroonDaily.Up.LastValue < aroonDaily.Down.LastValue)
    {
        //Do something
    }
}
```

StandardDeviation

Summary

The Standard Deviation indicator shows volatility.

Syntax

```
public StandardDeviation StandardDeviation(DataSeries source, int periods, MovingAverageType
maType)
```

Parameters

Name	Description

Example 1

```
[Parameter]
public DataSeries Source { get; set; }
[Parameter(DefaultValue = 20)]
public int Period { get; set; }
[Parameter("MA Type", DefaultValue = MovingAverageType.Simple)]
public MovingAverageType MAType { get; set; }
private StandardDeviation sd;
private double previousValue;
protected override void OnStart()
{
    sd = Indicators.StandardDeviation(Source, Period, MAType);
    previousValue = sd.Result.LastValue;
```

```
}
protected override void OnBar()
{
    //If StandardDeviation has increased
    if (sd.Result.LastValue > previousValue)
    {
        //Do something
    }
    //...
    previousValue = sd.Result.LastValue;
}
```

BollingerBands

Summary

The Bollinger Bands indicator shows volatility.

Syntax

```
public BollingerBands BollingerBands(DataSeries source, int periods, double
standardDeviations, MovingAverageType maType)
```

Parameters

Name	Description

Example 1

```
[Parameter]
public DataSeries Source { get; set; }
[Parameter(DefaultValue = 20)]
public int period { get; set; }
[Parameter("MA Type", DefaultValue = MovingAverageType.Simple)]
public MovingAverageType MAType { get; set; }
[Parameter(DefaultValue = 1.5)]
public double std { get; set; }
private BollingerBands bb;
protected override void OnStart()
{
    bb = Indicators.BollingerBands(Source, period, std, MAType);
}
protected override void OnTick()
{
    if (bb.Top.LastValue > Symbol.Bid)
```

```
    {
        Print("Bid price is higher than the Top Bollinger Band");
    }
}
```

RelativeStrengthIndex

Summary

The Relative Strength Index indicator measures turns in price by measuring turns in momentum.

Syntax

```
public RelativeStrengthIndex RelativeStrengthIndex(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
[Parameter]
public DataSeries Source { get; set; }
[Parameter(DefaultValue = 20)]
public int Period { get; set; }
private RelativeStrengthIndex rsi;
protected override void OnStart()
{
    rsi = Indicators.RelativeStrengthIndex(Source, Period);
}
protected override void OnBar()
{
    if (rsi.Result.LastValue > 70)
    {
        Print("RSI is higher than 70");
    }
}
```

TimeSeriesMovingAverage

Summary

The Time Series Moving Average is a moving average based on linear regression

Syntax

```
public TimeSeriesMovingAverage TimeSeriesMovingAverage(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
[Parameter]
public DataSeries Source { get; set; }
[Parameter(DefaultValue = 14)]
public int periodfast { get; set; }
[Parameter(DefaultValue = 24)]
public int periodslow { get; set; }
private TimeSeriesMovingAverage tsmfast;
private TimeSeriesMovingAverage tsmslow;
protected override void OnStart()
{
    tsmfast = Indicators.TimeSeriesMovingAverage(Source, periodfast);
    tsmslow = Indicators.TimeSeriesMovingAverage(Source, periodslow);
}
protected override void OnTick()
{
    //If TSMA with period 14 moves above TSMA with period 24
    if (tsmfast.Result.LastValue > tsmslow.Result.LastValue)
    {
        //Do something
    }
}
```

LinearRegressionForecast

Summary

Linear Regression Forecast is a trend indicator used to forecast values using the Least Squares Fit method.

Syntax

```
```

```
public LinearRegressionForecast LinearRegressionForecast(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
private LinearRegressionForecast lrForecast;
protected override void OnStart()
{
    lrForecast = Indicators.LinearRegressionForecast(Source, Period);
}
protected override void OnTick()
{
    Print("LRF Last Value = {0}", lrForecast.Result.LastValue);
}
```

LinearRegressionRSquared

Summary

The R Squared or coefficient of determination indicator's main purpose is the confirm the strength of the market.

Syntax

```
public LinearRegressionRSquared LinearRegressionRSquared(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
private LinearRegressionRSquared rSquared;
protected override void OnStart()
{
    rSquared = Indicators.LinearRegressionRSquared(Source, Period);
}
protected override void OnTick()
{
    Print("R squared is {0}", rSquared.Result.LastValue)
}
```

PriceROC

Summary

The Price Rate of Change indicator is the percentage change of the current price and the price N periods ago.

Syntax

```
public PriceROC PriceROC(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
[Parameter("Source")]
public DataSeries Source { get; set; }
[Parameter(DefaultValue = 14)]
public int Period { get; set; }
private PriceROC priceROC;
protected override void OnStart()
{
    priceROC = Indicators.PriceROC(Source, Period);
}
protected override void OnTick()
{
   Print("{0}", priceROC.Result.LastValue);
}
```

Vidya

Summary

Volatility Index Dynamic Average (VIDYA) is a smoothing (moving average) based on dynamically changing periods.

Syntax

```
public Vidya Vidya(DataSeries source, int periods, double r2Scale)
```

Parameters

Name	Description

Example 1

```
[Parameter]
public DataSeries Price { get; set; }
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
[Parameter("Sigma", DefaultValue = 0.65, MinValue = 0.1, MaxValue = 0.95)]
public double Sigma { get; set; }
private Vidya vidya;
protected override void OnStart()
{
    vidya = Indicators.Vidya(Price, Period, Sigma);
}
protected override void OnTick()
{
    //If vidya is greater than a specific value
    if (vidya.Result.LastValue > Value)
    {
        //Do something
        Print("LastValue {0}", vidya.Result.LastValue);
    }
    //...
}
```

UltimateOscillator

Summary

Returns the Ultimate Oscillator indicator instance.

Syntax

```
public UltimateOscillator UltimateOscillator(int cycle1, int cycle2, int cycle3)
```

```
public UltimateOscillator UltimateOscillator(MarketSeries marketSeries, int cycle1, int
cycle2, int cycle3)
```

Parameters

Name	Description

Example 1

```
protected override void OnStart()
{
   ultimateOscillator = Indicators.UltimateOscillator(Cycle1,Cycle2,Cycle3);
}
protected override void OnTick()
{
   double currentValue = ultimateOscillator.Result.LastValue;
    //...
}
```

UltimateOscillator

Summary

Initializes the UltimateOscillator instance for a specific timeframe

Syntax

```
public UltimateOscillator UltimateOscillator(int cycle1, int cycle2, int cycle3)
```

```
public UltimateOscillator UltimateOscillator(MarketSeries marketSeries, int cycle1, int
cycle2, int cycle3)
```

Parameters

Name	Description

DirectionalMovementSystem

Summary

The Directional Movement System is composed of three indicators that show if the market is trending and provide signals.

Syntax

```
```

```
public DirectionalMovementSystem DirectionalMovementSystem(int periods)
```

```
public DirectionalMovementSystem DirectionalMovementSystem(MarketSeries marketSeries, int
periods)
```

Parameters

Name	Description

Example 1

```
private DirectionalMovementSystem _dms;
private double _dIplus;
private double _dIminus;
[Parameter("ADX Period", DefaultValue = 14)]
public int Period { get; set; }

protected override void OnStart()
{
    _dms = Indicators.DirectionalMovementSystem(Period);
}
protected override void OnTick()
{
    _dIplus = _dms.DIPlus.LastValue;
    _dIminus = _dms.DIMinus.LastValue;
    if (_dIminus > _dIplus)
    {
        // Do something
    }
    //...
}
```

DirectionalMovementSystem

Summary

Initializes the Directional Movement System Indicator instance for a specific timeframe

Syntax

```
public DirectionalMovementSystem DirectionalMovementSystem(int periods)
```

```
```

```
public DirectionalMovementSystem DirectionalMovementSystem(MarketSeries marketSeries, int
periods)
```

Parameters

Name	Description

ParabolicSAR

Summary

The Parabolic SAR indicator identifies potential reversals in the market direction

Syntax

```
public ParabolicSAR ParabolicSAR(double minAf, double maxAf)
```

```
public ParabolicSAR ParabolicSAR(MarketSeries marketSeries, double minAf, double maxAf)
```

Parameters

Name	Description

Example 1

```
private ParabolicSAR parabolicSar;
//...
protected override void OnStart()
{
   parabolicSar = Indicators.ParabolicSAR(minaf, maxaf);
}
protected override void OnTick()
{
   double currentValue = parabolicSar.Result.LastValue;
    //...
}
```

ParabolicSAR

Summary

Initializes the ParabolicSAR Indicator instance for a specific timeframe

Syntax

```
public ParabolicSAR ParabolicSAR(double minAf, double maxAf)
```

```
public ParabolicSAR ParabolicSAR(MarketSeries marketSeries, double minAf, double maxAf)
```

Parameters

Name	Description

StochasticOscillator

Summary

The Stochastic Oscillator is a momentum indicator that aims to show price reversals by comparing the closing price to the price range.

Syntax

```
public StochasticOscillator StochasticOscillator(int kPeriods, int kSlowing, int dPeriods,
MovingAverageType maType)
```

```
public StochasticOscillator StochasticOscillator(MarketSeries marketSeries, int kPeriods, int
kSlowing, int dPeriods, MovingAverageType maType)
```

Parameters

Name	Description

Example 1

```
private StochasticOscillator stochastic;
//...
protected override void OnStart()
{
   stochastic = Indicators.StochasticOscillator(kPeriods, kSlowing, dPeriods, maType);
}
```

```
protected override void OnTick()
{
   double percentD = stochastic.PercentD.LastValue;
   double percentK = stochastic.PercentK.LastValue;
    //...
}
```

StochasticOscillator

Summary

Initializes the StochasticOscillator Indicator instance for a specific timeframe

Syntax

```
public StochasticOscillator StochasticOscillator(int kPeriods, int kSlowing, int dPeriods,
MovingAverageType maType)
```

```
public StochasticOscillator StochasticOscillator(MarketSeries marketSeries, int kPeriods, int
kSlowing, int dPeriods, MovingAverageType maType)
```

Parameters

Name	Description

MomentumOscillator

Summary

The Momentum Oscillator measures the momentum of the price.

Syntax

```
public MomentumOscillator MomentumOscillator(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
private MomentumOscillator _momentum;
protected override void OnStart()
{
    _momentum = Indicators.MomentumOscillator(MarketSeries.Close, 14);
}
protected override void OnTick()
{
    double momentum = _momentum.Result[index];
}
```

MedianPrice

Summary

The Median indicator is the average of the high and the low.

Syntax

```
public MedianPrice MedianPrice()
```

```
public MedianPrice MedianPrice(MarketSeries marketSeries)
```

Example 1

```
private MedianPrice medianPrice;
protected override void OnStart()
{
    medianPrice = Indicators.MedianPrice();
}
protected override void OnTick()
{
    double price = medianPrice.Result[index];
}
```

MedianPrice

Summary

Initializes the Median indicator instance for a specific timeframe

Syntax

```
public MedianPrice MedianPrice()
```

```
public MedianPrice MedianPrice(MarketSeries marketSeries)
```

Parameters

Name	Description

WilliamsAccumulationDistribution

Summary

The Williams Accumulation Distribution indicator shows bullish or bearish trends.

Syntax

```
public WilliamsAccumulationDistribution WilliamsAccumulationDistribution()
```

```
public WilliamsAccumulationDistribution WilliamsAccumulationDistribution(MarketSeries
marketSeries)
```

Example 1

```
private WilliamsAccumulationDistribution williamsAD;
protected override void OnStart()
{
    williamsAD = Indicators.WilliamsAccumulationDistribution();
}
protected override void OnTick()
{
    double result = williamsAD.Result[index];
}
```

WilliamsAccumulationDistribution

Summary

Initializes the WilliamsAccumulationDistribution indicator instance for a specific timeframe

Syntax

```
public WilliamsAccumulationDistribution WilliamsAccumulationDistribution()
```

```
public WilliamsAccumulationDistribution WilliamsAccumulationDistribution(MarketSeries
marketSeries)
```

Parameters

Name	Description

FractalChaosBands

Summary

The Fractal Chaos Bands indicator breaks down large trends into predictable patterns.

Syntax

```
public FractalChaosBands FractalChaosBands(int periods)
```

```
public FractalChaosBands FractalChaosBands(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

Example 1

```
private FractalChaosBands fractalChaosBands;
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
protected override void Initialize()
```

```
{
    fractalChaosBands = Indicators.FractalChaosBands(Period);
}
public override void Calculate(int index)
{
    Print("Fractal Chaos Bands High = {0}", fractalChaosBands.High[index]);
}
```

FractalChaosBands

Summary

Initializes the FractalChaosBands indicator instance for a specific timeframe

Syntax

```
public FractalChaosBands FractalChaosBands(int periods)
```

```
public FractalChaosBands FractalChaosBands(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

TypicalPrice

Summary

The Typical Price indicator is the average of the high, low, and closing prices.

Syntax

```
public TypicalPrice TypicalPrice()
```

```
public TypicalPrice TypicalPrice(MarketSeries marketSeries)
```

Example 1

```
private TypicalPrice typicalPriceIndicator;
protected override void Initialize()
{
    typicalPriceIndicator = Indicators.TypicalPrice();
}
public override void Calculate(int index)
{
    double typicalPriceValue = typicalPriceIndicator.Result[index];
}
```

TypicalPrice

Summary

Initializes the TypicalPrice indicator instance for a specific timeframe

Syntax

```
public TypicalPrice TypicalPrice()
```

```
public TypicalPrice TypicalPrice(MarketSeries marketSeries)
```

Parameters

Name	Description

CommodityChannelIndex

Summary

The Commodity Channel Index identifies overbough and oversold conditions, price reversals and trend strength.

Syntax

```
public CommodityChannelIndex CommodityChannelIndex(int periods)
```

```
public CommodityChannelIndex CommodityChannelIndex(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

Example 1

```
private CommodityChannelIndex commodityChannelIndex;
//...
protected override void OnStart()
{
    commodityChannelIndex = Indicators.CommodityChannelIndex(Periods);
}
protected override void OnBar()
{
    // Print the current value to the log
    Print("The current CCI value = {0}",
            commodityChannelIndex.Result.LastValue);
}
```

CommodityChannelIndex

Summary

Initializes the CommodityChannelIndex indicator instance for a specific timeframe

Syntax

```
public CommodityChannelIndex CommodityChannelIndex(int periods)
```

```
public CommodityChannelIndex CommodityChannelIndex(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

HistoricalVolatility

Summary

The Historical Volatility indicator is derived from time series of past market prices.

Syntax

```
public HistoricalVolatility HistoricalVolatility(DataSeries source, int periods, int
barHistory)
```

Parameters

Name	Description

Example 1

```
protected override void OnStart()
{
    historicalVolatility = Indicators.HistoricalVolatility
        (MarketSeries.Close, Period, BarHistory);
}
protected override void OnBar()
{
    double hv = historicalVolatility.Result.LastValue;
}
```

MassIndex

Summary

The Mass Index indicator is used to predict trend reversals.

Syntax

```
public MassIndex MassIndex(int periods)
```

```
public MassIndex MassIndex(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

Example 1

```
private MassIndex massIndex;
```

```
protected override void Initialize()
{
    massIndex = Indicators.MassIndex(14);
}
public override void Calculate(int index)
{
    double currentMassIndex = massIndex.Result[index];
}
```

MassIndex

Summary

Initializes the MassIndex indicator instance for a specific timeframe

Syntax

```
public MassIndex MassIndex(int periods)
```

```
public MassIndex MassIndex(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

ChaikinVolatility

Summary

The Chaikin Volatiliy indicator measures the trading range between the high and the low prices.

Syntax

```
public ChaikinVolatility ChaikinVolatility(int periods, int rateOfChange, MovingAverageType
maType)
```

```
public ChaikinVolatility ChaikinVolatility(MarketSeries marketSeries, int periods, int
rateOfChange, MovingAverageType maType)
```

Parameters

Name	Description

Example 1

```
private ChaikinVolatility chaikinVolatility;
protected override void OnStart()
{
    chaikinVolatility = Indicators.ChaikinVolatility(Periods, _roc, MaType);
}
protected override void OnBar()
{
    // Print to log
    Print("The Current Chaikin Volatility Value is: {0}",
              chaikinVolatility.Result.LastValue);
}
```

ChaikinVolatility

Summary

Initializes the ChaikinVolatility indicator instance for a specific timeframe

Syntax

```
public ChaikinVolatility ChaikinVolatility(int periods, int rateOfChange, MovingAverageType
maType)
```

```
public ChaikinVolatility ChaikinVolatility(MarketSeries marketSeries, int periods, int
rateOfChange, MovingAverageType maType)
```

Parameters

Name	Description

DetrendedPriceOscillator

Summary

The Detrended Price Oscillator shows intermediate overbought and oversold levels.

Syntax

```
public DetrendedPriceOscillator DetrendedPriceOscillator(DataSeries source, int periods,
MovingAverageType maType)
```

Parameters

Name	Description

Example 1

```
private _detrendedPriceOscillator _dpoFast;
private _detrendedPriceOscillator _dpoSlow;
protected override void OnStart()
{
    _dpoFast = Indicators.DetrendedPriceOscillator(Source, PeriodFast, MaType);
    _dpoSlow = Indicators.DetrendedPriceOscillator(Source, PeriodSlow, MaType);
}
protected override void OnBar()
{
    if(_dpoFast.Result.Count < 1)
        return;
    int currentIndex = _dpoFast.Result.Count - 1;
    int prevIndex = currentIndex - 1;
    if (_dpoFast.Result[prevIndex] > _dpoSlow.Result[prevIndex])
    {
        //Do something
    }
}
```

LinearRegressionIntercept

Summary

The Linear Regression Intercept can be used together with the Linear Regression Slope indicator to plot the Linear Regression Line.

Syntax

```
public LinearRegressionIntercept LinearRegressionIntercept(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
protected override void OnStart()
{
    // initialize a new instance of LinearRegressionIntercept indicator class
    _linearRegressionIntercept = Indicators.
        LinearRegressionIntercept(MarketSeries.Close, Period);
}
protected override void OnBar(int index)
{
    // Result of _linearRegressionIntercept at the current index
    double result = _linearRegressionIntercept.Result[index];
    // Print the current result to the log
    Print("Linear Regression Intercept at the current index is = {0}", result);
}
```

LinearRegressionSlope

Summary

The Linear Regression Slope indicator is intended to measure the direction and strength of a trend.

Syntax

```
public LinearRegressionSlope LinearRegressionSlope(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
private LinearRegressionSlope slope;
protected override void Initialize()
{
    slope = Indicators.LinearRegressionSlope(MarketSeries.Close, 14);
}
```

```
public override void Calculate(int index)
{
    double currentSlope = slope.Result[index];
}
```

MacdHistogram

Summary

The MACD Histogram is a momentum indicator measured by typically subtracting a 26 period moving average from a 12 period moving average.

Syntax

```
public MacdHistogram MacdHistogram(int longCycle, int shortCycle, int signalPeriods)
```

```
public MacdHistogram MacdHistogram(DataSeries source, int longCycle, int shortCycle, int
signalPeriods)
```

Parameters

Name	Description

Example 1

```
private MacdHistogram macd;
//...
protected override void Initialize()
{
    macd = Indicators.MacdHistogram(LongCycle, ShortCycle, Period);
    //...
}
public override void Calculate(int index)
{
    double macdHistogramResult = macd.Histogram[index];
    double macdSignalResult = macd.Signal[index];
    //...
}
```

MacdHistogram

Summary

Initializes the MacdHistogram indicator instance for a specific source series

Syntax

```
public MacdHistogram MacdHistogram(int longCycle, int shortCycle, int signalPeriods)
```

```
public MacdHistogram MacdHistogram(DataSeries source, int longCycle, int shortCycle, int
signalPeriods)
```

Parameters

Name	Description

MacdCrossOver

Summary

The MACD Line with the Signal line and their difference as a histogram.

Syntax

```
public MacdCrossOver MacdCrossOver(int longCycle, int shortCycle, int signalPeriods)
```

```
public MacdCrossOver MacdCrossOver(DataSeries source, int longCycle, int shortCycle, int
signalPeriods)
```

Parameters

Name	Description

Example 1

```
//...
private MacdCrossOver _macdCrossOver;

protected override void Initialize()
{
    _macdCrossOver = Indicators.MacdCrossOver(LongCycle, ShortCycle, Period);
```

```
}
public override void Calculate(int index)
{
    double macd = _macdCrossOver.MACD[index];
    double signal = _macdCrossOver.Signal[index];
    //...
}
```

MacdCrossOver

Summary

Initializes the MacdCrossOver indicator instance for a specific source series

Syntax

```
public MacdCrossOver MacdCrossOver(int longCycle, int shortCycle, int signalPeriods)
```

```
public MacdCrossOver MacdCrossOver(DataSeries source, int longCycle, int shortCycle, int
signalPeriods)
```

Parameters

Name	Description

PriceOscillator

Summary

The Price Oscillator calculates the difference between two moving averages.

Syntax

```
public PriceOscillator PriceOscillator(DataSeries source, int longCycle, int shortCycle,
MovingAverageType maType)
```

Parameters

Name	Description

Example 1

```
//...
private PriceOscillator priceOscillator;

protected override void OnStart()
{
    priceOscillator = Indicators.PriceOscillator
        (MarketSeries.Close, 14, 5, MovingAverageType.Simple);
    //...
}
protected override void OnTick()
{
    double result = priceOscillator.Result[index];
    //...
}
```

RainbowOscillator

Summary

The Rainbow Oscillator is a process of repetitive smoothing of simple moving averages resulting in a full spectrum of trends.

Syntax

```
public RainbowOscillator RainbowOscillator(DataSeries source, int levels, MovingAverageType
maType)
```

Parameters

Name	Description

Example 1

```
//...
private RainbowOscillator rainbow;
protected override void Initialize()
{
    MovingAverageType simpleMa = MovingAverageType.Simple;
    DataSeries close = MarketSeries.Close;
    rainbow = Indicators.RainbowOscillator(close, 9, simpleMa);
    //...
}
```

```
public override void Calculate(int index)
{
    double currentValue = rainbow.Result[index];
    //...
}
```

VerticalHorizontalFilter

Summary

The Vertical Horizontal Filter indicator measures the level of trend activity.

Syntax

```
public VerticalHorizontalFilter VerticalHorizontalFilter(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
//...
private VerticalHorizontalFilter VHFilter;
//...
protected override void Initialize()
{
    VHFilter = Indicators.VerticalHorizontalFilter(Source, Periods);
    //...
}
public override void Calculate(int index)
{
    double result = VHFilter.Result[index];
    //...
}
```

WilliamsPctR

Summary

The Williams Percent R indicator is a momentum indicator measuring overbought and oversold levels.

Syntax

```
public WilliamsPctR WilliamsPctR(int periods)
```

```
public WilliamsPctR WilliamsPctR(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

Example 1

```
//...
private WilliamsPctR williamsPctRSeries;
//...
protected override void OnStart()
{
    williamsPctRSeries = Indicators.WilliamsPctR(14);
    //...
}
protected override void OnTick()
{
    double williamsPctRValue = williamsPctRSeries.Result[index];
    //...
}
```

WilliamsPctR

Summary

Initializes the WilliamsPctR indicator instance for a specific timeframe

Syntax

```
public WilliamsPctR WilliamsPctR(int periods)
```

```
public WilliamsPctR WilliamsPctR(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

Trix

Summary

The Trix indicator shows the slope of a triple-smoothed exponential moving average.

Syntax

```
public Trix Trix(DataSeries source, int periods)
```

Parameters

Name	Description

Example 1

```
private Trix trixSeries;
protected override void OnStart()
{
    trixSeries = Indicators.Trix(MarketSeries.Close, 14);
}
protected override void OnTick()
{
    double trixValue = trixSeries.Result[index];
}
```

WeightedClose

Summary

The WeightedClose indicator is an average of each day's price with extra weight given to the closing price.

Remarks

Similar to the Median Price and Typical Price Indicators

Syntax

```
public WeightedClose WeightedClose()
```

```
public WeightedClose WeightedClose(MarketSeries marketSeries)
```

Example 1

```
//...
private WeightedClose weightedCloseSeries;
//...
protected override void OnStart()
{
    weightedCloseSeries = Indicators.WeightedClose();
    //...
}
protected override void OnBar()
{
    double weightedCloseValue = weightedCloseSeries.Result[index];
    //...
}
```

WeightedClose

Summary

Initializes the WeightedClose indicator instance for a specific timeframe

Syntax

```
public WeightedClose WeightedClose()
```

```
public WeightedClose WeightedClose(MarketSeries marketSeries)
```

Parameters

Name	Description

ChaikinMoneyFlow

Summary

The Chaikin Money Flow indicator measures the money flow volume over a specific period.

Syntax

```
public ChaikinMoneyFlow ChaikinMoneyFlow(int periods)
```

```
public ChaikinMoneyFlow ChaikinMoneyFlow(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

Example 1

```
private ChaikinMoneyFlow _chaikinMoneyFlow;
[Parameter("Period", DefaultValue = 21)]
public int Period { get; set; }
protected override void OnStart()
{
   _chaikinMoneyFlow = Indicators.ChaikinMoneyFlow(Period);
}
protected override void OnBar()
{
   var index = MarketSeries.Open.Count - 1;
   double currentChaikinMF = _chaikinMoneyFlow.Result[index];
   double previousChaikinMF = _chaikinMoneyFlow.Result[index-1];
}
```

ChaikinMoneyFlow

Summary

Initializes the ChaikinMoneyFlow indicator instance for a specific timeframe

Syntax

```
public ChaikinMoneyFlow ChaikinMoneyFlow(int periods)
```

```
public ChaikinMoneyFlow ChaikinMoneyFlow(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

EaseOfMovement

Summary

The Ease Of Movement indicator relates the price change to the volume.

Syntax

```
public EaseOfMovement EaseOfMovement(int periods, MovingAverageType maType)
```

```
public EaseOfMovement EaseOfMovement(MarketSeries marketSeries, int periods, MovingAverageType
maType)
```

Parameters

Name	Description

Example 1

```
private EaseOfMovement _easeOfMovement;
[Parameter("Period", DefaultValue = 14)]
public int Period { get; set; }
[Parameter("MA Type", DefaultValue = MovingAverageType.Simple)]
public MovingAverageType MAType { get; set; }
protected override void OnStart()
{
   _easeOfMovement = Indicators.EaseOfMovement(Period, MAType);
}
protected override void OnBar()
{
   // get EaseOfMovement value
   var index = MarketSeries.Open.Count - 1;
   double currentEaseOfMovement = _easeOfMovement.Result[index];
   double previousEaseOfMovement = _easeOfMovement.Result[index-1];
}
```

EaseOfMovement

Summary

Initializes the EaseOfMovement indicator instance for a specific timeframe

Syntax

```
public EaseOfMovement EaseOfMovement(int periods, MovingAverageType maType)
```

```
public EaseOfMovement EaseOfMovement(MarketSeries marketSeries, int periods, MovingAverageType
maType)
```

Parameters

Name	Description

MoneyFlowIndex

Summary

The Money Flow Index measures the strength of the money flow.

Syntax

```
public MoneyFlowIndex MoneyFlowIndex(int periods)
```

```
public MoneyFlowIndex MoneyFlowIndex(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

Example 1

```
private MoneyFlowIndex _moneyFlow;
[Parameter("Period", DefaultValue = 21)]
public int Period { get; set; }
protected override void OnStart()
{
    _moneyFlow = Indicators.MoneyFlowIndex(Period);
}
```

```
protected override void OnBar()
{
    var currentValue = _moneyFlow.Result.LastValue;
    //...
}
```

MoneyFlowIndex

Summary

Initializes the MoneyFlowIndex instance for a specific timeframe

Syntax

```
public MoneyFlowIndex MoneyFlowIndex(int periods)
```

```
public MoneyFlowIndex MoneyFlowIndex(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

NegativeVolumeIndex

Summary

The Negative Volume Index is a calculation of the percentage change in price on days when trading volume declines.

Syntax

```
public NegativeVolumeIndex NegativeVolumeIndex(DataSeries source)
```

Parameters

Name	Description

Example 1

```

```

```
private NegativeVolumeIndex _negativeVolume;
[Parameter]
public DataSeries Source { get; set; }
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
protected override void Initialize()
{
   _negativeVolume = Indicators.NegativeVolumeIndex(Source);
}
public override void Calculate(int index)
{
   // Display Result of Indicator
   Result[index] = _negativeVolume.Result[index];
}
```

OnBalanceVolume

Summary

The On Balance Volume indicator relates price and volume.

Syntax

```
public OnBalanceVolume OnBalanceVolume(DataSeries source)
```

Parameters

Name	Description

PositiveVolumeIndex

Summary

The Positive Volume Index is a calculation of the percentage change in price on days when trading volume increased.

Syntax

```
public PositiveVolumeIndex PositiveVolumeIndex(DataSeries source)
```

Parameters

Name	Description

Example 1

```
private PositiveVolumeIndex _positiveVolume;
[Parameter]
public DataSeries Source { get; set; }
protected override void OnStart()
{
    _positiveVolume = Indicators.PositiveVolumeIndex(Source);
}
protected override void OnBar()
{
    var currentValue = _positiveVolume.Result.LastValue;
    //...
}
```

PriceVolumeTrend

Summary

The Price Volume Trend indicator shows the relationship between price and volume.

Syntax

```
public PriceVolumeTrend PriceVolumeTrend(DataSeries source)
```

Parameters

Name	Description

Example 1

```
private PriceVolumeTrend _priceVolumeTrend;
[Parameter]
public DataSeries Source { get; set; }
protected override void OnStart()
{
    _priceVolumeTrend = Indicators.PriceVolumeTrend(Source);
}
protected override void OnBar()
{
    var currentValue = _priceVolumeTrend.Result.LastValue;
```

```
    //...
}
```

TradeVolumeIndex

Summary

Trade Volume Index indicator measures the amount of money flowing in and out of an asset.

Syntax

```
public TradeVolumeIndex TradeVolumeIndex(DataSeries source)
```

Parameters

Name	Description

Example 1

```
private TradeVolumeIndex _tradeVolume;
[Parameter]
public DataSeries Source { get; set; }
protected override void OnStart()
{
    _tradeVolume = Indicators.TradeVolumeIndex(Source);
}
protected override void OnBar()
{
    var currentValue = _tradeVolume.Result.LastValue;
    //...
}
```

VolumeOscillator

Summary

The Volume Oscillator indicator is the difference between two moving averages.

Syntax

```

```

```
public VolumeOscillator VolumeOscillator(int shortTerm, int longTerm)
```

```
public VolumeOscillator VolumeOscillator(MarketSeries marketSeries, int shortTerm, int
longTerm)
```

Parameters

Name	Description

Example 1

```
private VolumeOscillator _volumeOscillator;
[Parameter("Short Term", DefaultValue = 9)]
public int ShortTerm { get; set; }
[Parameter("Long Term", DefaultValue = 21)]
public int LongTerm { get; set; }
protected override void OnStart()
{
    _volumeOscillator = Indicators.VolumeOscillator(ShortTerm, LongTerm);;
}
protected override void OnBar()
{
    var currentValue = _volumeOscillator.Result.LastValue;
    //...
}
```

VolumeOscillator

Summary

Initializes the VolumeOscillator instance for a specific timeframe

Syntax

```
public VolumeOscillator VolumeOscillator(int shortTerm, int longTerm)
```

```
public VolumeOscillator VolumeOscillator(MarketSeries marketSeries, int shortTerm, int
longTerm)
```

Parameters

Name	Description

VolumeROC

Summary

Volume Rate of Change Indicator measures the rate of change of the tick volume.

Syntax

```
public VolumeROC VolumeROC(int periods)
```

```
public VolumeROC VolumeROC(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

Example 1

```
//...
private VolumeROC _volumeROC;
//...
[Parameter("Period", DefaultValue = 21)]
public int Period { get; set; }
protected override void OnStart()
{
    _volumeROC = Indicators.VolumeROC(Period);
}
protected override void OnBar()
{
    var currentValue = _volumeROC.Result.LastValue;
    //...
}
```

VolumeROC

Summary

Initializes the VolumeROC instance for a specific timeframe

Syntax

```
public VolumeROC VolumeROC(int periods)
```

```
public VolumeROC VolumeROC(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

AverageTrueRange

Summary

Average true range. An indicator providing the degree of price volatility.

Remarks

Average true range (ATR) is a technical analysis volatility indicator originally developed by J. Welles Wilder. The indicator provides the degree of price volatility. The average true range is an N-day exponential moving average of the true range values. Wilder recommended a 14-period smoothing.

Syntax

```
public AverageTrueRange AverageTrueRange(int periods, MovingAverageType maType)
```

```
public AverageTrueRange AverageTrueRange(MarketSeries marketSeries, int periods,
MovingAverageType maType)
```

Parameters

Name	Description

Example 1

```
private AverageTrueRange atrIndicator;
[Parameter(DefaultValue = 20)]
public int Period { get; set; }
[Parameter("MA Type", DefaultValue = MovingAverageType.Simple)]
public MovingAverageType MAType { get; set; }
[Parameter(DefaultValue = 0.002)]
```

```
public double ATRValue { get; set; }
protected override void OnStart()
{
    atrIndicator = Indicators.AverageTrueRange(Period, MAType);
}
protected override void OnTick()
{
    //If atrIndicator last value is greater than the ATRValue input
    if (atrIndicator.Result.LastValue > ATRValue)
    {
        // Do something
    }
    //...
}
```

AverageTrueRange

Summary

Initializes the AverageTrueRange instance for a specific timeframe

Syntax

```
public AverageTrueRange AverageTrueRange(int periods, MovingAverageType maType)
```

```
public AverageTrueRange AverageTrueRange(MarketSeries marketSeries, int periods,
MovingAverageType maType)
```

Parameters

Name	Description

DonchianChannel

Summary

The Donchian channel is a volatility indicator forming a channel between the highest high and the lowest low of the chosen period.

Remarks

The Donchian channel is mainly used for providing entry signals. A long is established when the price closes above the

Donchian Channel. Conversely, if it closes below, then a short is established.

Syntax

```
public DonchianChannel DonchianChannel(int periods)
```

```
public DonchianChannel DonchianChannel(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

Example 1

```
//...
private DonchianChannel donchian;
//...
protected override void OnStart()
{
    donchian = Indicators.DonchianChannel(Period);
}
protected override void OnBar()
{
    Print("Top Value = {0}", donchian.Top.LastValue);
    Print("Middle Value = {0}", donchian.Middle.LastValue);
    Print("Bottom Value = {0}", donchian.Bottom.LastValue);
    //...
}
```

DonchianChannel

Summary

Initializes the DonchianChannel instance for a specific timeframe

Syntax

```
public DonchianChannel DonchianChannel(int periods)
```

```
public DonchianChannel DonchianChannel(MarketSeries marketSeries, int periods)
```

Parameters

Name	Description

IchimokuKinkoHyo

Summary

Ichimoku Kinko Hyo Indicator is a moving average based trend identification system.

Remarks

Ichimoku Kinko Hyo Indicator contains more data points than standard candlestick charts and thus provides a clearer picture of potential price action.

Syntax

```
public IchimokuKinkoHyo IchimokuKinkoHyo(int tenkanSenPeriods, int kijunSenPeriods, int
senkouSpanBPeriods)
```

```
public IchimokuKinkoHyo IchimokuKinkoHyo(MarketSeries marketSeries, int tenkanSenPeriods, int
kijunSenPeriods, int senkouSpanBPeriods)
```

Parameters

Name	Description

Example 1

```
//...
private IchimokuKinkoHyo ichimokuKinkoHyo;
//...
protected override void OnStart()
{
    ichimokuKinkoHyo = Indicators.IchimokuKinkoHyo
        (tenkanSenPeriods, kijunSenPeriods, senkouSpanBPeriods);
}
protected override void OnBar()
{
    Print("ChikouSpan Value = {0}", ichimokuKinkoHyo.ChikouSpan.LastValue);
    Print("KijunSen Value = {0}", ichimokuKinkoHyo.KijunSen.LastValue);
    Print("SenkouSpanA Value = {0}", ichimokuKinkoHyo.SenkouSpanA.LastValue);
    Print("SenkouSpanB Value = {0}", ichimokuKinkoHyo.SenkouSpanB.LastValue);
    Print("TenkanSen Value = {0}", ichimokuKinkoHyo.TenkanSen.LastValue);
```

```
    //...
}
```

IchimokuKinkoHyo

Summary

Initializes the IchimokuKinkoHyo indicator instance for a specific timeframe

Syntax

```
public IchimokuKinkoHyo IchimokuKinkoHyo(int tenkanSenPeriods, int kijunSenPeriods, int senkouSpanBPeriods)
```

```
public IchimokuKinkoHyo IchimokuKinkoHyo(MarketSeries marketSeries, int tenkanSenPeriods, int kijunSenPeriods, int senkouSpanBPeriods)
```

Parameters

Name	Description

AwesomeOscillator

Summary

Initializes the AwesomeOscillator indicator instance

Syntax

```
public AwesomeOscillator AwesomeOscillator()
```

```
public AwesomeOscillator AwesomeOscillator(MarketSeries marketSeries)
```

AcceleratorOscillator

Summary

Initializes the AcceleratorOscillator indicator instance

Syntax

```
public AcceleratorOscillator AcceleratorOscillator()
```

```
public AcceleratorOscillator AcceleratorOscillator(MarketSeries marketSeries)
```

AwesomeOscillator

Summary

Initializes the AwesomeOscillator instance for specific timeframe

Syntax

```
public AwesomeOscillator AwesomeOscillator()
```

```
public AwesomeOscillator AwesomeOscillator(MarketSeries marketSeries)
```

Parameters

Name	Description

AcceleratorOscillator

Summary

Initializes the AcceleratorOscillator instance for specific timeframe

Syntax

```
public AcceleratorOscillator AcceleratorOscillator()
```

```
public AcceleratorOscillator AcceleratorOscillator(MarketSeries marketSeries)
```

Parameters

Name	Description

KeltnerChannels

Summary

Initializes the Keltner Channels indicator instance

Syntax

```
public KeltnerChannels KeltnerChannels(int maPeriod, MovingAverageType maType, int atrPeriod,
MovingAverageType atrMaType, double bandDistance)
```

```
public KeltnerChannels KeltnerChannels(MarketSeries marketSeries, int maPeriod,
MovingAverageType maType, int atrPeriod, MovingAverageType atrMaType, double bandDistance)
```

Parameters

Name	Description

KeltnerChannels

Summary

Initializes the Keltner Channels indicator instance for specific MarketSeries

Syntax

```
public KeltnerChannels KeltnerChannels(int maPeriod, MovingAverageType maType, int atrPeriod,
MovingAverageType atrMaType, double bandDistance)
```

```
public KeltnerChannels KeltnerChannels(MarketSeries marketSeries, int maPeriod,
MovingAverageType maType, int atrPeriod, MovingAverageType atrMaType, double bandDistance)
```

Parameters

Name	Description

INotifications

Summary

It is an interface that represents all Notifications.

Syntax

```
public interface INotifications
```

Members

Name	Type	Summary
PlaySound	Method	Plays a notification sound.
SendEmail	Method	Sends a notification email message.

PlaySound

Summary

Plays a notification sound.

Remarks

In indicators, use it with IsRealTime/IsLastBar, for real-time values.

Syntax

```
public void PlaySound(string fileName)
```

Parameters

Name	Description

Example 1

```
Notifications.PlaySound(@"C:\SampleDestination\SampleSound.mp3");
```

SendEmail

Summary

Sends a notification email message.

Remarks

Use correct settings before trying to send an email notification. You can do that in Preferences -> Email Settings

Syntax

```
public void SendEmail(string from, string to, string subject, string text)
```

Parameters

Name	Description

Example 1

```
Notifications.SendEmail("from@email.com", "to@email.com",
            "Email Notification Subject", "Email body");
```

IServer

Summary

Server related information.

Syntax

```
public interface IServer
```

Members

Name	Type	Summary
Connected	Event	Event raised when successfully connected with the server
Disconnected (2)	Event	Disconnected - Event raised when connection with the server is lost

IsConnected	Property	Indicates current connection status with the server
Time (5)	Property	Returns the server time.

Time

Summary

Returns the server time.

Syntax

```
public DateTime Time{ get; }
```

Example 1

```
protected override void OnTick()
{
    Print("The Server Time is: {0}", Server.Time);
}
```

IsConnected

Summary

Indicates current connection status with the server

Syntax

```
public bool IsConnected{ get; }
```

Connected

Summary

Event raised when successfully connected with the server

Syntax

```
public event Action Connected
```

Disconnected

Summary

Disconnected - Event raised when connection with the server is lost

Syntax

```
public event Action Disconnected
```

ISmallScriptsController

Syntax

```
public interface ISmallScriptsController
```

Members

Name	Type	Summary
RefreshData (2)	Method	

RefreshData

Syntax

```
public void RefreshData()
```

LeverageTier

Summary

Tier of dynamic leverage.

Syntax

```
public interface LeverageTier
```

Members

Name	Type	Summary
Leverage	Property	Leverage of dynamic leverage tier.
Volume	Property	Volume of dynamic leverage tier.

Example 1

```
var firstTier = Symbol.DynamicLeverage[0];
Print("Leverage for volume up to {0} is {1}, firstTier.Volume, firstTier.Leverage);
```

Volume

Summary

Volume of dynamic leverage tier.

Syntax

```
public double Volume{ get; }
```

Leverage

Summary

Leverage of dynamic leverage tier.

Syntax

```

```

```
public double Leverage{ get; }
```

MarketData

Summary

Provides access to Depth of Market Data

Syntax

```
public interface MarketData
```

Members

Name	Type	Summary
GetMarketDepth	Method	Get the depth of market prices and volumes of the symbol passed as parameter
GetSeries	Method	Get the MarketSeries of a specific timeframe and the current symbol
GetSymbol	Method	Get the Symbol given the symbol's string name representation

Example 1

```
MarketDepth md = MarketData.GetMarketDepth(Symbol);
```

GetMarketDepth

Summary

Get the depth of market prices and volumes of the symbol passed as parameter

Syntax

```
public MarketDepth GetMarketDepth(string symbolCode)
```

```
public MarketDepth GetMarketDepth(Symbol symbol)
```

Parameters

Name	Description

Example 1

```
MarketDepth md = MarketData.GetMarketDepth("EURUSD");
```

GetMarketDepth

Summary

Get the depth of market price and volume of the current symbol

Syntax

```
public MarketDepth GetMarketDepth(string symbolCode)
```

```
public MarketDepth GetMarketDepth(Symbol symbol)
```

Parameters

Name	Description

Example 1

```
MarketDepth md = MarketData.GetMarketDepth(Symbol);
```

GetSeries

Summary

Get the MarketSeries of a specific timeframe and the current symbol

Syntax

```
public MarketSeries GetSeries(TimeFrame timeFrame)
```

```
public MarketSeries GetSeries(string symbolCode, TimeFrame timeFrame)
```

```
public MarketSeries GetSeries(Symbol symbol, TimeFrame timeFrame)
```

Parameters

Name	Description

Example 1

```
var marketSeriesMin30 = MarketData.GetSeries(TimeFrame.Minute30);
var smaMin30 = Indicators.SimpleMovingAverage(marketSeriesMin30.Close, 14);
```

GetSeries

Summary

Get the MarketSeries of market data for the symbol and timeframe

Syntax

```
public MarketSeries GetSeries(TimeFrame timeFrame)
```

```
public MarketSeries GetSeries(string symbolCode, TimeFrame timeFrame)
```

```
public MarketSeries GetSeries(Symbol symbol, TimeFrame timeFrame)
```

Parameters

Name	Description

Example 1

```
var daily = MarketData.GetSeries("EURUSD", TimeFrame.Daily);
var sma = Indicators.SimpleMovingAverage(daily.Close, 14);
```

GetSeries

Summary

Get the MataSeries of market data for the symbol and timeframe

Syntax

```
public MarketSeries GetSeries(TimeFrame timeFrame)
```

```
public MarketSeries GetSeries(string symbolCode, TimeFrame timeFrame)
```

```
public MarketSeries GetSeries(Symbol symbol, TimeFrame timeFrame)
```

Parameters

Name	Description

Example 1

```
Symbol USDCAD = GetSymbol("USDCAD");
var daily = MarketData.GetSeries(USDCAD, TimeFrame.Daily);
var sma = Indicators.SimpleMovingAverage(daily.Close, 14);
```

GetSymbol

Summary

Get the Symbol given the symbol's string name representation

Syntax

```
public Symbol GetSymbol(string symbolCode)
```

Parameters

Name	Description

Example 1

```
Symbol USDCAD = MarketData.GetSymbol("USDCAD");
var usdCadAsk = USDCAD.Ask;
```

MarketHours

Summary

Access to symbol's trading sessions schedule

Syntax

```
public interface MarketHours
```

Members

Name	Type	Summary
IsOpened	Method	Indicates if trading session is active
Sessions	Property	List of all symbol's trading sessions
TimeTillClose	Method	Time left till end of current trading session. Returns 0 if session is not active
TimeTillOpen	Method	Time left till start of new trading session. Returns 0 if session is already active

Sessions

Summary

List of all symbol's trading sessions

Syntax

```
public IReadonlyList Sessions{ get; }
```

IsOpened

Summary

Indicates if trading session is active

Syntax

```
public bool IsOpened()
```

```
public bool IsOpened(DateTime datetime)
```

IsOpened

Summary

Indicates if trading session is active

Syntax

```
public bool IsOpened()
```

```
public bool IsOpened(DateTime datetime)
```

Parameters

Name	Description

TimeTillClose

Summary

Time left till end of current trading session. Returns 0 if session is not active

Syntax

```
```

```
public TimeSpan TimeTillClose()
```

TimeTillOpen

Summary

Time left till start of new trading session. Returns 0 if session is already active

Syntax

```
public TimeSpan TimeTillOpen()
```

MarketSeries

Summary

Provides access to the market data such as the DataSeries Open, High, Low, Close.

Remarks

Access to the Open, Hight, Low, Close, Median, Typical, Weighted price series as well as OpenTime for the current symbol and time frame.

Syntax

```
public interface MarketSeries
```

Members

Name	Type	Summary
Close	Property	Close price series of historical trendbars.
High (2)	Property	Highest price series of historical trendbars.
Low (2)	Property	Low price series of historical trendbars.
Median	Property	Median price series of historical trendbars.
Open	Property	Open price series of historical trendbars.
OpenTime	Property	Open Time series of historical trendbars.
SymbolCode (2)	Property	The code representation of the symbol that the MarketSeries is subscribed to

TickVolume (2)	Property	Volume of Ticks for Historical Trendbars
TimeFrame (3)	Property	The timeframe that the MarketSeries is subscribed to
Typical	Property	Typical price series of historical trendbars.
WeightedClose (2)	Property	Weighted price series of historical trendbars.

Example 1

```
//Accessing historical OHLC prices from Indicators
double close = MarketSeries.Close[index];
double high = MarketSeries.High[index];
double low = MarketSeries.Low[index];
double open = MarketSeries.Open[index];
```

Example 2

```
//Accessing historical O-H-L-C prices from Robots
int index = MarketSeries.Close.Count-1;
double close = MarketSeries.Close[index];
double high = MarketSeries.High[index];
double low = MarketSeries.Low[index];
double open = MarketSeries.Open[index];
```

Open

Summary

Open price series of historical trendbars.

Syntax

```
public DataSeries Open{ get; }
```

Example 1

```
[Parameter(DefaultValue = 14)]
public int period { get; set; }
private SimpleMovingAverage smaopen
protected override void Initialize()
{
    //Simple moving average of the Open price series for the specified period
```

```
    smaopen = Indicators.SimpleMovingAverage(MarketSeries.Open, period);
}
```

High

Summary

Highest price series of historical trendbars.

Syntax

```
public DataSeries High{ get; }
```

Example 1

```
[Parameter(DefaultValue = 14)]
public int period { get; set; }
private SimpleMovingAverage smahigh
protected override void Initialize()
{
    //Simple moving average of the High price series for the specified period
    smahigh = Indicators.SimpleMovingAverage(MarketSeries.High,period);
}
```

Low

Summary

Low price series of historical trendbars.

Syntax

```
public DataSeries Low{ get; }
```

Example 1

```
[Parameter(DefaultValue = 14)]
public int period { get; set; }
```

```
private SimpleMovingAverage smaLow
protected override void Initialize()
{
    //Simple moving average of the Low price series for the specified period
    smaLow = Indicators.SimpleMovingAverage(MarketSeries.Low, period);
}
```

Close

Summary

Close price series of historical trendbars.

Syntax

```
public DataSeries Close{ get; }
```

Example 1

```
[Parameter(DefaultValue = 14)]
public int Period { get; set; }
private SimpleMovingAverage smaClose
protected override void Initialize()
{
    //Simple moving average of the Close prices series for the specified period
    smaClose = Indicators.SimpleMovingAverage(MarketSeries.Close, Period);
}
```

TickVolume

Summary

Volume of Ticks for Historical Trendbars

Syntax

```
public DataSeries TickVolume{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    double currentVolume = MarketSeries.TickVolume[index];
    double previousVolume = MarketSeries.TickVolume[index-1];
}
```

Median

Summary

Median price series of historical trendbars.

Syntax

```
public DataSeries Median{ get; }
```

Example 1

```
protected override void OnBar()
{
    int currentIndex = MarketSeries.Median.Count - 1;
    var currentTypical = MarketSeries.Median[currentIndex];     //The current Median price
    var previousTypical = MarketSeries.Median[currentIndex - 1]; //The previous Median price
    if (currentTypical < previousTypical)
    {
        // Do something
    }
}
```

Typical

Summary

Typical price series of historical trendbars.

Syntax

```
public DataSeries Typical{ get; }
```

Example 1

```
public override void Calculate(int index)
{
    if (IsRealTime)
    {
        var currentTypical = MarketSeries.Typical[index];
        var previousTypical = MarketSeries.Typical[index - 1];
        if (currentTypical < previousTypical)
        {
            Print("Current typical price is less than the previous one");
        }
    }
}
```

WeightedClose

Summary

Weighted price series of historical trendbars.

Syntax

```
public DataSeries WeightedClose{ get; }
```

Example 1

```
protected override void OnBar()
{
    int currentIndex = MarketSeries.Weighted.Count - 1;
    var currentWeighted = MarketSeries.Weighted[currentIndex];     //The current Weighted
price
    var previousWeighted = MarketSeries.Weighted[currentIndex - 1]; //The previous Weighted
price
    if (currentWeighted <= previousWeighted)
    {
        // Do something
    }
}
```

OpenTime

Summary

Open Time series of historical trendbars.

Syntax

```
public TimeSeries OpenTime{ get; }
```

Example 1

```
//Accessing historical Open Times
Print("{0}", MarketSeries.OpenTime[index]);
Print("{0}", MarketSeries.OpenTime[index].Day);
Print("{0}", MarketSeries.OpenTime[index].Hour);
```

TimeFrame

Summary

The timeframe that the MarketSeries is subscribed to

Syntax

```
public TimeFrame TimeFrame{ get; }
```

Example 1

```
Print("{0}", MarketSeries.TimeFrame);
```

Example 2

```
Symbol eurUsd = MarketData.GetSymbol("EURUSD");
MarketSeries seriesEurUsd = MarketData.GetSeries(eurUsd, TimeFrame.Daily);
Print(seriesEurUsd.TimeFrame);
```

SymbolCode

Summary

The code representation of the symbol that the MarketSeries is subscribed to

Syntax

```
public string SymbolCode{ get; }
```

Example 1

```
Print("{0}", MarketSeries.SymbolCode);
```

Example 2

```
Symbol eurUsd = MarketData.GetSymbol("EURUSD");
MarketSeries seriesEurUsd = MarketData.GetSeries(eurUsd, TimeFrame.Daily);
Print(seriesEurUsd.SymbolCode);
```

Symbol

Summary

Represents a currency pair

Syntax

```
public interface Symbol
```

Members

Name	Type	Summary
Ask	Property	The current ask price for this symbol.
Bid	Property	The current bid price for this symbol.
Code (2)	Property	Represents the trading pair code, e.g. "EURUSD".
Digits	Property	The number of digits for the symbol.
DynamicLeverage	Property	Dynamic leverage tiers for symbol.
LotSize	Property	Size of 1 lot in units of the base currency.

MarketHours	Property	Access to the symbol's trading sessions schedule.
NormalizeVolumeInUnits	Method	Round the volume to the amount suitable for a trade.
PipSize	Property	Pip size for current symbol.
PipValue	Property	The monetary value of one pip.
QuantityToVolumeInUnits	Method	Convert Quantity (in lots) to Volume in units of base currency.
Spread	Property	The current spread of this symbol.
TickSize	Property	Tick size for the current symbol.
TickValue	Property	The monetary value of one tick.
UnrealizedGrossProfit (2)	Property	Sum of the unrealized Gross profits of the positions of this Symbol
UnrealizedNetProfit (2)	Property	Sum of the unrealized Net profits of the positions of this Symbol.
VolumeInUnitsMax	Property	The maximum tradable amount.
VolumeInUnitsMin	Property	The minimum tradable amount.
VolumeInUnitsStep	Property	The minimum trade amount increment.
VolumeInUnitsToQuantity	Method	Convert Volume in units of base currency to Quantity (in lots).

Example 1

```
double bid = Symbol.Bid;
double ask = Symbol.Ask;
string code = Symbol.Code;
int digits = Symbol.Digits;
double pipSize = Symbol.PipSize;
double pointSize = Symbol.PointSize;
double spread = Symbol.Spread;
```

Ask

Summary

The current ask price for this symbol.

Remarks

The seller's price for the symbol.

Syntax

```
public double Ask{ get; }
```

Example 1

```
protected override void OnTick()
{
    Print("Ask Price: {0}", Symbol.Ask);
}
```

Bid

Summary

The current bid price for this symbol.

Remarks

The buyer's price for the symbol.

Syntax

```
public double Bid{ get; }
```

Example 1

```
protected override void OnTick()
{
    Print("Bid Price: {0}", Symbol.Bid);
}
```

Spread

Summary

The current spread of this symbol.

Remarks

The difference between the Ask and the Bid price for the symbol. (see Ask and Bid)

Syntax

```
public double Spread{ get; }
```

Example 1

```
protected override void OnTick()
{
    Print("The Spread of the symbol is: {0}", Symbol.Spread);
}
```

Code

Summary

Represents the trading pair code, e.g. "EURUSD".

Syntax

```
public string Code{ get; }
```

Example 1

```
protected override void OnTick()
{
    Print("This strategy is trading the symbol: {0}", Symbol.Code);
}
```

PipSize

Summary

Pip size for current symbol.

Syntax

```
public double PipSize{ get; }
```

Example 1

```
protected override void OnTick()
{
    Print("The current symbol has pip size of: {0}", Symbol.PipSize);
}
```

Digits

Summary

The number of digits for the symbol.

Syntax

```
public int Digits{ get; }
```

Example 1

```
protected override void OnTick()
{
    Print("The number of Digits the current symbol  has is: {0}", Symbol.Digits);
}
```

TickSize

Summary

Tick size for the current symbol.

Remarks

If the symbol is a 5 digit symbol, the tick size is 0.00001.

Syntax

```
public double TickSize{ get; }
```

Example 1

```
protected override void OnTick()
{
    Print("The current symbol has TickSize: {0}", Symbol.TickSize);
}
```

VolumeInUnitsMin

Summary

The minimum tradable amount.

Syntax

```
public double VolumeInUnitsMin{ get; }
```

Example 1

```
if(volume < Symbol.VolumeInUnitsMin)
{
    Print("The minimum volume is {0}", Symbol.VolumeInUnitsMin);
}
```

VolumeInUnitsMax

Summary

The maximum tradable amount.

Syntax

```
public double VolumeInUnitsMax{ get; }
```

Example 1

```
if(Symbol.NormalizeVolumeInUnits(volume, RoundingMode.Down) <= Symbol.VolumeInUnitsMax)
```

```
{
    volume = Symbol.NormalizeVolumeInUnits(volume, RoundingMode.Down);
    ExecuteMarketOrder(TradeType.Buy, Symbol, volume);
}
```

VolumeInUnitsStep

Summary

The minimum trade amount increment.

Syntax

```
public double VolumeInUnitsStep{ get; }
```

Example 1

```
if(volume + Symbol.VolumeInUnitsStep <= Symbol.VolumeInUnitsMax)
{
    volume += Symbol.VolumeInUnitsStep;
}
```

PipValue

Summary

The monetary value of one pip.

Syntax

```
public double PipValue{ get; }
```

Example 1

```
var volume = ((Account.Balance*Risk)/StopLoss)/Symbol.PipValue;
```

TickValue

Summary

The monetary value of one tick.

Syntax

```
public double TickValue{ get; }
```

Example 1

```
var volume = ((Account.Balance*Risk)/StopLoss)/Symbol.TickValue;
```

LotSize

Summary

Size of 1 lot in units of the base currency.

Syntax

```
public long LotSize{ get; }
```

UnrealizedNetProfit

Summary

Sum of the unrealized Net profits of the positions of this Symbol.

Syntax

```
public double UnrealizedNetProfit{ get; }
```

UnrealizedGrossProfit

Summary

Sum of the unrealized Gross profits of the positions of this Symbol

Syntax

```
public double UnrealizedGrossProfit{ get; }
```

DynamicLeverage

Summary

Dynamic leverage tiers for symbol.

Syntax

```
public IReadonlyList DynamicLeverage{ get; }
```

Example 1

```
var symbolLeverage = Symbol.DynamicLeverage[0];
var realLeverage = Math.Min(symbolLeverage, Account.Leverage);
```

MarketHours

Summary

Access to the symbol's trading sessions schedule.

Syntax

```
public MarketHours MarketHours{ get; }
```

NormalizeVolumeInUnits

Summary

Round the volume to the amount suitable for a trade.

Syntax

```
public double NormalizeVolumeInUnits(double volume, [optional] RoundingMode roundingMode)
```

Parameters

Name	Description

Example 1

```
volume = Symbol.NormalizeVolumeInUnits(volume, RoundingMode.Down);
```

QuantityToVolumeInUnits

Summary

Convert Quantity (in lots) to Volume in units of base currency.

Syntax

```
public double QuantityToVolumeInUnits(double quantity)
```

Parameters

Name	Description

VolumeInUnitsToQuantity

Summary

Convert Volume in units of base currency to Quantity (in lots).

Syntax

```
public double VolumeInUnitsToQuantity(double volume)
```

Parameters

Name	Description

TradingSession

Summary

Trading session schedule

Syntax

```
public interface TradingSession
```

Members

Name	Type	Summary
EndDay	Property	Day of week when trading session ends
EndTime	Property	Time when trading session ends
StartDay	Property	Day of week when trading session starts
StartTime	Property	Time when trading session starts

StartDay

Summary

Day of week when trading session starts

Syntax

```
public DayOfWeek StartDay{ get; }
```

EndDay

Summary

Day of week when trading session ends

Syntax

```
public DayOfWeek EndDay{ get; }
```

StartTime

Summary

Time when trading session starts

Syntax

```
public TimeSpan StartTime{ get; }
```

EndTime

Summary

Time when trading session ends

Syntax

```
public TimeSpan EndTime{ get; }
```

LevelsAttribute

Summary

Levels Attribute. Applies metadata to enable the plot of level lines.

Remarks

Represents level lines. It is commonly used in Oscillators, for instance to add a zero line. Must be added before the indicator class declaration.

Syntax

```
public sealed class LevelsAttribute : Attribute
```

Members

Name	Type	Summary
Levels	Property	The array of price values that are ploted as level lines
LevelsAttribute	Method	Initializes a new LevelsAttribute instance

Example 1

```
namespace cAlgo.Indicators
{
    [Levels(0, 50, 100)]
    [Indicator()]
    public class NewIndicator : Indicator
    //...
```

Levels

Summary

The array of price values that are ploted as level lines

Syntax

```
public Double[] Levels{ get; set; }
```

Example 1

```
namespace cAlgo.Indicators
{
    // two level lines will be drawn at prices 20.0 and 80.5
    [Levels(20.0, 80.5)]
    [Indicator]
```

```
    public class NewIndicator : Indicator
    //...
```

LevelsAttribute

Summary

Initializes a new LevelsAttribute instance

Remarks

Draws level (horizontal) lines at a fixed position when the plot is on the indicator panel below the chart. To make it effective apply enclosed in square brackets, e.g. [Levels(0)], before the Indicator atribute declaration.

Syntax

```
public LevelsAttribute LevelsAttribute(Double[] levels)
```

Parameters

Name	Description

Example 1

```
namespace cAlgo.Indicators
{
    // A zero line will be drawn
    [Levels(0)]
    [Indicator]
    public class NewIndicator : Indicator
```

LineStyle

Summary

An enumeration of different stroke styles used to render lines.

Syntax

```
public sealed enum LineStyle
```

Members

Name	Type	Summary
Dots (2)	Field	A dotted line: .....
DotsRare	Field	A dotted line, large gap between dots: . . . .
DotsVeryRare	Field	A dotted line, extra large gap between dots: . . . .
Lines	Field	Lines with gaps are used to render the line: - - - -
LinesDots	Field	A mixed line / dot style is used to render the line: - . - . - .
Solid	Field	A solid line: -----

Example 1

```
//Examples of all different LineStyles
[Output("Dots", LineStyle = LineStyle.Dots)]
public IndicatorDataSeries outputDots { get; set; }
[Output("DotsRare", LineStyle = LineStyle.DotsRare)]
public IndicatorDataSeries outputDotsRare { get; set; }
[Output("DotsVeryRare", LineStyle = LineStyle.DotsVeryRare)]
public IndicatorDataSeries outputDotsVeryRare { get; set; }
[Output("Lines", LineStyle = LineStyle.Lines)]
public IndicatorDataSeries outputLines { get; set; }
[Output("LinesDots", LineStyle = LineStyle.LinesDots)]
public IndicatorDataSeries outputLinesDots { get; set; }
[Output("Solid", LineStyle = LineStyle.Solid)]
public IndicatorDataSeries outputSolid { get; set; }
```

Solid

Summary

A solid line: -----

Syntax

```
LineStyle.Solid
```

Example 1

```
[Output("Solid", LineStyle = LineStyle.Solid)]
```

```
public IndicatorDataSeries outputSolid { get; set; }
```

Dots

Summary

A dotted line: .....

Syntax

```
LineStyle.Dots
```

Example 1

```
[Output("Dots", LineStyle = LineStyle.Dots)]
public IndicatorDataSeries outputDots { get; set; }
```

DotsRare

Summary

A dotted line, large gap between dots: . . . .

Syntax

```
LineStyle.DotsRare
```

Example 1

```
[Output("DotsRare", LineStyle = LineStyle.DotsRare)]
public IndicatorDataSeries outputDotsRare { get; set; }
```

DotsVeryRare

Summary

A dotted line, extra large gap between dots: . . . .

Syntax

```
LineStyle.DotsVeryRare
```

Example 1

```
[Output("DotsVeryRare", LineStyle = LineStyle.DotsVeryRare)]
public IndicatorDataSeries outputDotsVeryRare { get; set; }
```

LinesDots

Summary

A mixed line / dot style is used to render the line: - . - . - .

Syntax

```
LineStyle.LinesDots
```

Example 1

```
[Output("LinesDots", LineStyle = LineStyle.LinesDots)]
public IndicatorDataSeries outputLinesDots { get; set; }
```

Lines

Summary

Lines with gaps are used to render the line: - - - -

Syntax

```
LineStyle.Lines
```

Example 1

```
[Output("Lines", LineStyle = LineStyle.Lines)]
public IndicatorDataSeries outputLines { get; set; }
```

MarketDepth

Summary

Access to MarketDepth Ask Entries, Bid Entries and the event at which the market depth gets updated

Syntax

```
public interface MarketDepth
```

Members

Name	Type	Summary
AskEntries	Property	The total number of Ask entries
BidEntries	Property	The total number of Bid entries
Updated	Event	The event at which the market depth gets updated

Example 1

```
using System;
using System.Text;
using cAlgo.API;
namespace cAlgo.Indicators
{
    [Indicator]
    public class MarketDepthIndicator : Indicator
    {
        private MarketDepth _marketDepth;
        public override void Calculate(int index){}
        protected override void Initialize()
        {
            //  Get Market Depth
            _marketDepth = MarketData.GetMarketDepth(Symbol);
            // subscribe to event Updated
            _marketDepth.Updated += MarketDepthUpdated;
        }
        void MarketDepthUpdated()
```

```
        {
            // Draw Market Depth Entries in the indicator panel
            var se = new StringBuilder();
            se.Append("Bid");
            se.Append("                                ");
            se.Append("Ask");
            ChartObjects.DrawText("DOM", se.ToString(), StaticPosition.TopLeft, Colors.White);
            se.Clear();
            se.AppendLine();
            se.AppendLine();
            foreach (var entry in _marketDepth.BidEntries)
            {
                double dVolume  = Math.Round(entry.Volume / 1000000.0, 2);
                string volume = string.Format("{0}{1}", dVolume, "m");
                double entryPrice = entry.Price;
                string askText = string.Format("{0}    {1}", entryPrice.ToString("0.00000"),
volume);
                se.AppendLine(askText);
            }
            ChartObjects.DrawText("Bid", se.ToString(), StaticPosition.TopLeft, Colors.Red);

            se.Clear();
            se.AppendLine();
            se.AppendLine();
            foreach (var entry in _marketDepth.AskEntries)
            {
                double dVolume = Math.Round(entry.Volume / 1000000.0, 2);
                string volume = string.Format("{0}{1}", dVolume, "m");
                double entryPrice = entry.Price;
                se.Append("                                        ");
                string bidText = string.Format("{0}     {1}", entryPrice.ToString("0.00000"),
volume);
                se.AppendLine(bidText);
            }
            ChartObjects.DrawText("Ask", se.ToString(), StaticPosition.TopLeft,
Colors.Turquoise);

        }
    }
}
```

Example 2

```
using cAlgo.API;
namespace cAlgo.Indicators
{
    [Indicator]
    public class Level2 : Indicator
    {
        [Output("BidEntries", Color = Colors.Red, PlotType = PlotType.Histogram, Thickness =
5)]
```

```
        public IndicatorDataSeries BidResult { get; set; }
        [Output("AskEntries", Color = Colors.Blue, PlotType = PlotType.Histogram, Thickness =
5)]
        public IndicatorDataSeries AskResult { get; set; }
        MarketDepth GBPUSD;
        private int _askNo;
        private int _bidNo;
        protected override void Initialize()
        {
            GBPUSD = MarketData.GetMarketDepth(Symbol);
            GBPUSD.Updated += OnGbpUsdUpdated;
        }
        void OnGbpUsdUpdated()
        {
            _askNo = 0;
            _bidNo = 0;
            var index = MarketSeries.Close.Count - 1;
            for (var i = 0; i < GBPUSD.AskEntries.Count; i++)
                AskResult[index - i] = double.NaN;
            foreach (var entry in GBPUSD.AskEntries)
            {
                AskResult[index - _askNo] = (-1) * entry.Volume;
                _askNo++;
            }
            for (var i = 0; i < GBPUSD.BidEntries.Count; i++)
                BidResult[index - i] = double.NaN;
            foreach (var entry in GBPUSD.BidEntries)
            {
                BidResult[index - _bidNo] = entry.Volume;
                _bidNo++;
            }
        }
        public override void Calculate(int index){}
    }
}
```

AskEntries

Summary

The total number of Ask entries

Syntax

```
public IReadonlyList AskEntries{ get; }
```

Example 1

```
foreach (var entry in _marketDepth.AskEntries)
{
     volume  = entry.Volume;
            entryPrice = entry.Price;
}
```

BidEntries

Summary

The total number of Bid entries

Syntax

```
public IReadonlyList BidEntries{ get; }
```

Example 1

```
foreach (var entry in _marketDepth.BidEntries)
{
     volume  = entry.Volume;
            entryPrice = entry.Price;
}
```

Updated

Summary

The event at which the market depth gets updated

Syntax

```
public event Action Updated
```

Example 1

```
MarketDepth _marketDepth;
protected override void Initialize()
{
    _marketDepth = MarketData.GetMarketDepth(Symbol);
    // subscribe to event Updated
    _marketDepth.Updated += MarketDepthUpdated;
}
// user defined function MarketDepthUpdated
void MarketDepthUpdated()
{
    // Do something
}
```

MarketDepthEntry

Summary

Provides access to market depth prices and volumes

Syntax

```
public interface MarketDepthEntry
```

Members

Name	Type	Summary
Price	Property	The price of this market depth entry
VolumeInUnits (2)	Property	The volume of this market depth entry

Example 1

```
foreach (var marketDepthEntry in _marketDepth.AskEntries)
{
     //The volume of this market depth entry
     volume = marketDepthEntry.Volume;
     //The price of this market depth entry
     price = marketDepthEntry.Price;
}
```

VolumeInUnits

Summary

The volume of this market depth entry

Syntax

```
public double VolumeInUnits{ get; }
```

Example 1

```
foreach (var entry in _marketDepth.AskEntries)
{
     volume  = entry.VolumeInUnits;
}
```

Price

Summary

The price of this market depth entry

Syntax

```
public double Price{ get; }
```

Example 1

```
for(int i = 0; i < _marketDepth.AskEntries.Count; i++)
{
    price = _marketDepth.AskEntries[i].Price;
}
```

MovingAverageType

Summary

An enumeration of the different MovingAverage weighting (smoothing) methods.

Syntax

```
public sealed enum MovingAverageType
```

Members

Name	Type	Summary
Exponential	Field	Use exponential weighting. Represents indicator of ExponentialMovingAverage type.
Simple	Field	Use uniform weighting. Represents indicator of SimpleMovingAverage type.
TimeSeries	Field	Represents indicator of TimeSeriesMovingAverage type.
Triangular	Field	Represents indicator of TriangularMovingAverage type.
VIDYA	Field	VIDYA (Volatility Index Dynamic Average) variable length weighting. Represents indicator of Vidya type.
Weighted	Field	Represents indicator of WeightedMovingAverage type.
WilderSmoothing	Field	Represents indicator of WellesWilderSmoothing type.

Simple

Summary

Use uniform weighting. Represents indicator of SimpleMovingAverage type.

Syntax

```
MovingAverageType.Simple
```

Example 1

```
[Parameter("MAType", DefaultValue = MovingAverageType.Simple)]
public MovingAverageType MaType { get; set; }
```

Exponential

Summary

Use exponential weighting. Represents indicator of ExponentialMovingAverage type.

Syntax

```
MovingAverageType.Exponential
```

Example 1

```
[Parameter("MAType", DefaultValue = MovingAverageType.Exponential)]
public MovingAverageType MaType { get; set; }
```

TimeSeries

Summary

Represents indicator of TimeSeriesMovingAverage type.

Syntax

```
MovingAverageType.TimeSeries
```

Example 1

```
[Parameter("MAType", DefaultValue = MovingAverageType.TimeSeries)]
public MovingAverageType MaType { get; set; }
```

Triangular

Summary

Represents indicator of TriangularMovingAverage type.

Syntax

```
MovingAverageType.Triangular
```

Example 1

```
[Parameter("MAType", DefaultValue = MovingAverageType.Triangular)]
public MovingAverageType MaType { get; set; }
```

VIDYA

Summary

VIDYA (Volatility Index Dynamic Average) variable length weighting. Represents indicator of Vidya type.

Syntax

```
MovingAverageType.VIDYA
```

Example 1

```
[Parameter("MAType", DefaultValue = MovingAverageType.VIDYA)]
public MovingAverageType MaType { get; set; }
```

Weighted

Summary

Represents indicator of WeightedMovingAverage type.

Syntax

```
MovingAverageType.Weighted
```

Example 1

```
[Parameter("MAType", DefaultValue = MovingAverageType.Weighted)]
public MovingAverageType MaType { get; set; }
```

WilderSmoothing

Summary

Represents indicator of WellesWilderSmoothing type.

Syntax

```
MovingAverageType.WilderSmoothing
```

Example 1

```
[Parameter("MAType", DefaultValue = MovingAverageType.WilderSmoothing)]
public MovingAverageType MaType { get; set; }
```

Example 2

```
private MovingAverageType _wilderSmoothing = MovingAverageType.WilderSmoothing;
```

OutputAttribute

Summary

Sealed Class OutputAttribute

Remarks

Marks a IndicatorDataSeries property as output to be displayed on the chart or panel below. To make it effective please apply this attribute in front of the declaration of the IndicatorDataSeries to be displayed.

Syntax

```
public sealed class OutputAttribute : Attribute
```

Members

Name	Type	Summary
IsHistogram	Property	Plots a Histogram.
LineColor	Property	Gets or sets the Color of the Output property. This Color will be used when the line for this Output is plotted.
LineStyle (8)	Property	Gets or sets the Line Style for given Output property. By default it is set to Solid

Name (3)	Property	The plot name
OutputAttribute	Method	Initializes a new instance of the OutputAttribute and sets the name.
PlotType	Property	Plot type.
Thickness (8)	Property	Sets the Width of the Output property.

LineStyle

Summary

Gets or sets the Line Style for given Output property. By default it is set to Solid

Remarks

If PlotType = PlotType.Line (default) the LineStyle can be added. Supported line styles are: Dots DotsRare DotsVeryRare Lines LinesDots Solid

Syntax

```
public LineStyle LineStyle{ get; set; }
```

Example 1

```
//...
//Simple moving average will be now plotted as Lines.
[Output("Simple Moving Average", LineStyle = LineStyle.Lines)]
public IndicatorDataSeries SMA { get; set; }

//...
```

Name

Summary

The plot name

Remarks

Displayed in the User Interface when adding an new instance of the Indicator.

Syntax

```
public string Name{ get; }
```

Example 1

```
//...
//The plotted indicator name is Simple Moving Average.
[Output("Simple Moving Average")]
public IndicatorDataSeries SMA { get; set; }
//...
```

LineColor

Summary

Gets or sets the Color of the Output property. This Color will be used when the line for this Output is plotted.

Syntax

```
public string LineColor{ get; set; }
```

Example 1

```
//...
//The result is plotted in Turquoise.
[Output("Main", LineColor = "#008000")]
public IndicatorDataSeries SMA { get; set; }
public override void Calculate(int index)
{
   //...
}
```

Thickness

Summary

Sets the Width of the Output property.

Remarks

This Width will be used when the line for this Output is plotted.

Syntax

```
public float Thickness{ get; set; }
```

Example 1

```
//...
//The result is plotted as a line with thickness level five
[Output("Simple Moving Average", Thickness = 5)]
public IndicatorDataSeries SMA { get; set; }
public override void Calculate(int index)
{
    //...
}
```

IsHistogram

Summary

Plots a Histogram.

Syntax

```
public bool IsHistogram{ get; set; }
```

Example 1

```
[Output("Main", IsHistogram = true)]
public IndicatorDataSeries Result { get; set; }
```

PlotType

Summary

Plot type.

Remarks

The type of the output plotted on the output panel. Default = Line Supported types are: Line Points Histogram

Syntax

```
public PlotType PlotType{ get; set; }
```

Example 1

```
//...
//The result is plotted as a Histogram.
[Output("Commodity Channel Index", PlotType = PlotType.Histogram)]
public IndicatorDataSeries SMA { get; set; }
public override void Calculate(int index)
{
    //...
}
```

Example 2

```
//...
//Plot the result as a set of yellow points.
[Output("Main", LineColor = "Yellow", PlotType = PlotType.Points)]
public IndicatorDataSeries Result { get; set; }
//...
```

OutputAttribute

Summary

Initializes a new instance of the OutputAttribute and sets the name.

Remarks

The members have the following default values: PlotType = PlotType.Line; LineStyle = LineStyle.Solid; Thickness = 1f; LineColor = "Green"; Name = lineName;

Syntax

```
public OutputAttribute OutputAttribute(string lineName)
```

Parameters

Name	Description

Example 1

```
// Simple plot that uses the default solid line plot in green color
[Output("Main")]
public IndicatorDataSeries Result { get; set; }
```

Example 2

```
//Plot a simple moving average with a set of lines (dashes)
[Output("Simple Moving Average", LineStyle = LineStyle.Lines)]
public IndicatorDataSeries SMA { get; set; }
```

Example 3

```
//...
//Plot a Histogram.
[Output("Commodity Channel Index", PlotType = PlotType.Histogram)]
public IndicatorDataSeries Result { get; set; }
//...
```

ParameterAttribute

Summary

Class ParameterAttribute

Remarks

Marks a property as input parameter

Syntax

```
public class ParameterAttribute : Attribute
```

Members

Name	Type	Summary

DefaultValue	Property	Gets or sets the default value of this Parameter property.
MaxValue	Property	Gets or sets the maximum value of this Parameter property. It is used for validating user input.
MinValue	Property	Gets or sets the minimum value of this Parameter property. It is used for validating user input.
Name (4)	Property	The input parameter name.
ParameterAttribute	Method	Initializes a new ParameterAttribute instance and sets the name
Step	Property	Gets or sets the step of this Parameter. Step is used in NumericUpDown controls in parameter editors.

Name

Summary

The input parameter name.

Syntax

```
public string Name{ get; }
```

Example 1

```
//...
//The input parameter name is MaPeriod
[Parameter("MaPeriod")]
public int Period { get; set; }
//...
```

DefaultValue

Summary

Gets or sets the default value of this Parameter property.

Syntax

```
public Object DefaultValue{ get; set; }
```

Example 1

```
//...
//The value for Periods is fourteen
[Parameter(DefaultValue = 14)]
public int Periods { get; set; }
//...
```

MinValue

Summary

Gets or sets the minimum value of this Parameter property. It is used for validating user input.

Syntax

```
public Object MinValue{ get; set; }
```

Example 1

```
//...
//The minimum value the user can set Periods is five.
[Parameter(DefaultValue = 14, MinValue = 5)]
public int Periods { get; set; }
//...
```

MaxValue

Summary

Gets or sets the maximum value of this Parameter property. It is used for validating user input.

Syntax

```
public Object MaxValue{ get; set; }
```

Example 1

```
//...
//The maximum value the user can set Periods is twenty
[Parameter(DefaultValue = 14, MaxValue = 20)]
public int Periods { get; set; }
//...
```

Step

Summary

Gets or sets the step of this Parameter. Step is used in NumericUpDown controls in parameter editors.

Syntax

```
public double Step{ get; set; }
```

ParameterAttribute

Summary

Initializes a new ParameterAttribute instance and sets the name

Remarks

Represents an input parameter to the program. To make it effective type in enclosed in square brackets, e.g. [Parameter], before the property declaration. Parameters are listed in the instance button of the robot/indicator.

Syntax

```
public ParameterAttribute ParameterAttribute(string name)
```

```
public ParameterAttribute ParameterAttribute()
```

Parameters

Name	Description

Example 1

```
//...
// parameter attribute
[Parameter("Parameter Name")]
public int ParameterName { get; set; }
//...
```

Example 2

```
//...
// parameter attribute
[Parameter("ParameterName", DefaultValue = 14, MinValue = 2, MaxValue = 30)]
public int ParameterName { get; set; } //  property declaration
//...
```

ParameterAttribute

Summary

Initializes a new instance of the ParameterAttribute class.

Remarks

In this case the parameter name is the same as the property name.

Syntax

```
public ParameterAttribute ParameterAttribute(string name)
```

```
public ParameterAttribute ParameterAttribute()
```

Example 1

```
//...
// The Parameter name is MaPeriod
[Parameter]
public int MaPeriod { get; set; }
//...
```

PendingOrder

Summary

Provides access to properties of pending orders

Syntax

```
public interface PendingOrder
```

Members

Name	Type	Summary
Cancel	Method	Shortcut for Robot.CancelPendingOrder method
Comment (3)	Property	User assigned Order Comment
ExpirationTime	Property	The order Expiration time The Timezone used is set in the Robot attribute
HasTrailingStop	Property	When HasTrailingStop set to true, server updates Stop Loss every time position moves in your favor.
Id	Property	Unique order Id.
Label (2)	Property	User assigned identifier for the order.
ModifyExpirationTime	Method	Shortcut for Robot.ModifyPendingOrder method to change Expiration Time
ModifyStopLimitRange	Method	Shortcut for Robot.ModifyPendingOrder method to change Stop Limit Range
ModifyStopLossPips	Method	Shortcut for Robot.ModifyPendingOrder method to change Stop Loss
ModifyTakeProfitPips	Method	Shortcut for Robot.ModifyPendingOrder method to change Take Profit
ModifyTargetPrice	Method	Shortcut for Robot.ModifyPendingOrder method to change Target Price
ModifyVolume	Method	Shortcut for Robot.ModifyPendingOrder method to change VolumeInUnits
OrderType	Property	Specifies whether this order is Stop or Limit.
Quantity (2)	Property	Quantity (lots) of this order
StopLimitRangePips	Property	Maximum limit from order target price, where order can be executed.
StopLoss	Property	The order stop loss in price
StopLossPips	Property	The order stop loss in pips
StopLossTriggerMethod	Property	Trigger method for position's StopLoss
StopOrderTriggerMethod	Property	Determines how pending order will be triggered in case it's a StopOrder
SymbolCode (3)	Property	Symbol code of the order
TakeProfit	Property	The order take profit in price
TakeProfitPips	Property	The order take profit in pips

TargetPrice	Property	The order target price.
TradeType (2)	Property	Specifies whether this order is to buy or sell.
VolumeInUnits (3)	Property	Volume of this order.

Example 1

```
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,Symbol.Bid);
var order = LastResult.PendingOrder;
Print("The pending order's ID: {0}", order.Id);
```

SymbolCode

Summary

Symbol code of the order

Syntax

```
public string SymbolCode{ get; }
```

Example 1

```
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,Symbol.Bid);
Print("SymbolCode = {0}", LastResult.PendingOrder.SymbolCode);
```

TradeType

Summary

Specifies whether this order is to buy or sell.

Syntax

```
public TradeType TradeType{ get; }
```

Example 1

```
PlaceLimitOrder(TradeType.Buy, Symbol, 10000, targetPrice);
Print(LastResult.PendingOrder.TradeType);
```

VolumeInUnits

Summary

Volume of this order.

Syntax

```
public double VolumeInUnits{ get; }
```

Example 1

```
var result = PlaceLimitOrder(TradeType.Buy, Symbol, 10000, targetPrice);
var order = result.PendingOrder;
Print("The order's volume is: {0}", order.VolumeInUnits);
```

Id

Summary

Unique order Id.

Syntax

```
public int Id{ get; }
```

Example 1

```
var result = PlaceLimitOrder(TradeType.Buy, Symbol, 10000, targetPrice);
var order = result.PendingOrder;
Print("The pending order's ID: {0}", order.Id);
```

OrderType

Summary

Specifies whether this order is Stop or Limit.

Syntax

```
public PendingOrderType OrderType{ get; }
```

Example 1

```
var result = PlaceLimitOrder(TradeType.Buy, Symbol, 10000, targetPrice);
var order = result.PendingOrder;
Print("Order type = {0}", order.OrderType);
```

TargetPrice

Summary

The order target price.

Syntax

```
public double TargetPrice{ get; }
```

Example 1

```
var targetPrice = Symbol.Bid;
var result = PlaceLimitOrder(TradeType.Buy, Symbol, 10000, targetPrice);
```

ExpirationTime

Summary

The order Expiration time The Timezone used is set in the Robot attribute

Syntax

```
public DateTime? ExpirationTime{ get; }
```

Example 1

```
DateTime expiration = Server.Time.AddMinutes(120);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
    Symbol.Bid, null, 10, 10, expiration);
```

StopLoss

Summary

The order stop loss in price

Syntax

```
public double? StopLoss{ get; }
```

Example 1

```
var result = PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
Symbol.Bid, null, 10, 10);
var order = result.PendingOrder;
Print("Order SL price = {0}", order.StopLoss);
```

StopLossPips

Summary

The order stop loss in pips

Syntax

```
public double? StopLossPips{ get; }
```

Example 1

```
var result = PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                    Symbol.Bid, null, 10, 10);
var order = result.PendingOrder;
Print("Order SL pips = {0}", order.StopLossPips);
```

TakeProfit

Summary

The order take profit in price

Syntax

```
public double? TakeProfit{ get; }
```

Example 1

```
var result = PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
Symbol.Bid, null, 10, 10);
var order = result.PendingOrder;
Print("Order TP price = {0}", order.TakeProfit);
```

TakeProfitPips

Summary

The order take profit in pips

Syntax

```
public double? TakeProfitPips{ get; }
```

Example 1

```
var result = PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
```

```
Symbol.Bid, null, 10, 10);
var order = result.PendingOrder;
Print("TP Pips = {0}", order.TakeProfitPips);
```

Label

Summary

User assigned identifier for the order.

Syntax

```
public string Label{ get; }
```

Example 1

```
var result = PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
Symbol.Bid, "myLabel", 10, 10);
if(result.IsSuccessful)
{
    var order = result.PendingOrder;
    Print("Label = {0}", order.Label);
}
```

Comment

Summary

User assigned Order Comment

Syntax

```
public string Comment{ get; }
```

Example 1

```
var result = PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                Symbol.Bid, null, 10, 10, null, "this is a comment");
```

```
var order = result.PendingOrder;
Print("comment = {0}", order.Comment);
```

Quantity

Summary

Quantity (lots) of this order

Syntax

```
public double Quantity{ get; }
```

HasTrailingStop

Summary

When HasTrailingStop set to true, server updates Stop Loss every time position moves in your favor.

Syntax

```
public bool HasTrailingStop{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10, 10, 2, "comment", true);
Print("Position was opened, has Trailing Stop = {0}", result.Position.HasTrailingStop);
```

StopLossTriggerMethod

Summary

Trigger method for position's StopLoss

Syntax

```
public StopTriggerMethod? StopLossTriggerMethod{ get; }
```

StopOrderTriggerMethod

Summary

Determines how pending order will be triggered in case it's a StopOrder

Syntax

```
public StopTriggerMethod? StopOrderTriggerMethod{ get; }
```

StopLimitRangePips

Summary

Maximum limit from order target price, where order can be executed.

Syntax

```
public double? StopLimitRangePips{ get; }
```

Example 1

```
var targetPrice = Symbol.Ask;
var result = PlaceStopLimitOrder(TradeType.Buy, Symbol, 10000, targetPrice, 2.0);
```

ModifyStopLossPips

Summary

Shortcut for Robot.ModifyPendingOrder method to change Stop Loss

Syntax

```
public TradeResult ModifyStopLossPips(double? stopLossPips)
```

Parameters

Name	Description

ModifyTakeProfitPips

Summary

Shortcut for Robot.ModifyPendingOrder method to change Take Profit

Syntax

```
public TradeResult ModifyTakeProfitPips(double? takeProfitPips)
```

Parameters

Name	Description

ModifyStopLimitRange

Summary

Shortcut for Robot.ModifyPendingOrder method to change Stop Limit Range

Syntax

```
public TradeResult ModifyStopLimitRange(double stopLimitRangePips)
```

Parameters

Name	Description

ModifyExpirationTime

Summary

Shortcut for Robot.ModifyPendingOrder method to change Expiration Time

Syntax

```
public TradeResult ModifyExpirationTime(DateTime? expirationTime)
```

Parameters

Name	Description

ModifyVolume

Summary

Shortcut for Robot.ModifyPendingOrder method to change VolumeInUnits

Syntax

```
public TradeResult ModifyVolume(double volume)
```

Parameters

Name	Description

ModifyTargetPrice

Summary

Shortcut for Robot.ModifyPendingOrder method to change Target Price

Syntax

```
public TradeResult ModifyTargetPrice(double targetPrice)
```

Parameters

Name	Description

Cancel

Summary

Shortcut for Robot.CancelPendingOrder method

Syntax

```
public TradeResult Cancel()
```

PendingOrderCancellationReason

Summary

Reason for order cancellation

Syntax

```
public sealed enum PendingOrderCancellationReason
```

Members

Name	Type	Summary
Cancelled	Field	Order was cancelled by trader
Expired	Field	Order was cancelled due to expiration
Rejected	Field	Order fill was rejected and order was cancelled

Cancelled

Summary

Order was cancelled by trader

Syntax

```
PendingOrderCancellationReason.Cancelled
```

Expired

Summary

Order was cancelled due to expiration

Syntax

```
PendingOrderCancellationReason.Expired
```

Rejected

Summary

Order fill was rejected and order was cancelled

Syntax

```
PendingOrderCancellationReason.Rejected
```

PendingOrderCancelledEventArgs

Summary

Provides data for the pending order cancellation event.

Syntax

```
public class PendingOrderCancelledEventArgs : Object
```

Members

Name	Type	Summary
PendingOrder	Property	Gets the pending order that was cancelled.
Reason	Property	Gets the reason for the pending order cancellation.

Example 1

```
protected override void OnStart()
{
    PendingOrders.Cancelled += PendingOrdersOnCancelled;
    var result = PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask + 10 *
Symbol.PipSize);
    CancelPendingOrder(result.PendingOrder);
}
private void PendingOrdersOnCancelled(PendingOrderCancelledEventArgs args)
{
   Print("Pending order with id {0} was cancelled. Reason: {1}", args.PendingOrder.Id,
args.Reason);
}
```

PendingOrder

Summary

Gets the pending order that was cancelled.

Syntax

```
public PendingOrder PendingOrder{ get; }
```

Reason

Summary

Gets the reason for the pending order cancellation.

Syntax

```
public PendingOrderCancellationReason Reason{ get; }
```

PendingOrderCreatedEventArgs

Summary

Provides data for the pending order creation events.

Syntax

```
public class PendingOrderCreatedEventArgs : Object
```

Members

Name	Type	Summary
PendingOrder (2)	Property	Gets the pending order that was created.

Example 1

```
protected override void OnStart()
{
    PendingOrders.Created+= PendingOrdersOnCreated;
    PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask + 10 * Symbol.PipSize)
}
private void PendingOrdersOnCreated(PendingOrderCreatedEventArgsargs)
{
    Print("Pending order with id {0} was created", args.PendingOrder.Id);
}
```

PendingOrder

Summary

Gets the pending order that was created.

Syntax

```
public PendingOrder PendingOrder{ get; }
```

PendingOrderFilledEventArgs

Summary

Provides data for the pending order fill event.

Syntax

```
public class PendingOrderFilledEventArgs : Object
```

Members

Name	Type	Summary
PendingOrder (3)	Property	Gets the pending order that was filled.
Position	Property	Gets the position that was filled from the pending order.

Example 1

```
protected override void OnStart()
{
    PendingOrders.Filled += PendingOrdersOnFilled;
    PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask);
}
private void PendingOrdersOnFilled(PendingOrderFilledEventArgs args)
{
    Print("Pending order with id {0} was filled, position id  is {1}", args.PendngOrder.Id,
args.Position.Id);
}
```

PendingOrder

Summary

Gets the pending order that was filled.

Syntax

```
public PendingOrder PendingOrder{ get; }
```

Position

Summary

Gets the position that was filled from the pending order.

Syntax

```
public Position Position{ get; }
```

PendingOrderModifiedEventArgs

Summary

Provides data for the pending order modification event.

Syntax

```
public class PendingOrderModifiedEventArgs : Object
```

Members

Name	Type	Summary
PendingOrder (4)	Property	Gets the pending order that was modified.

Example 1

```
protected override void OnStart()
{
    PendingOrders.Modified += PendingOrdersOnModified;
    var result = PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask + 10 *
Symbol.PipSize);
    ModifyPendingOrder(result.PendingOrder, Symbol.Ask + 20 * Symbol.PipSize ,null, null,
null);
}
private void PendingOrdersOnModified(PendingOrderModifiedEventArgs args)
{
    Print("Pending order with id {0} was modifed", args.PendingOrder.Id);
}
```

PendingOrder

Summary

Gets the pending order that was modified.

Syntax

```
public PendingOrder PendingOrder{ get; }
```

PendingOrders

Summary

Provides access to methods of the Pending Orders collection

Syntax

```
public interface PendingOrders : IEnumerable
```

Members

Name	Type	Summary
Cancelled (2)	Event	Occurs when pending order is cancelled
Count (4)	Property	Total number of pending orders
Created	Event	Occurs when pending order is created
Filled	Event	Occurs when pending order is filled
Modified	Event	Occurs when pending order is modified
this[int index] (5)	Property	Find a pending order by index

this[int index]

Summary

Find a pending order by index

Syntax

```
public PendingOrder this[int index]{ get; }
```

Parameters

Name	Description

Example 1

```
if(PendingOrders.Count > 0)
    Print(PendingOrders[0].Id);
```

Count

Summary

Total number of pending orders

Syntax

```
public int Count{ get; }
```

Example 1

```
var totalOrders = PendingOrders.Count;
```

Created

Summary

Occurs when pending order is created

Syntax

```
public event Action Created
```

Example 1

```
protected override void OnStart()
{
    PendingOrders.Created += PendingOrdersOnCreated;
```

```
    PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask + 10 * Symbol.PipSize);
}
private void PendingOrdersOnCreated(PendingOrderCreatedEventArgs args)
{
    Print("Pending order with id {0} was created", args.PendingOrder.Id);
}
```

Modified

Summary

Occurs when pending order is modified

Syntax

```
public event Action Modified
```

Example 1

```
protected override void OnStart()
{
    PendingOrders.Modified += PendingOrdersOnModified;
    var result = PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask + 10 *
Symbol.PipSize);
    ModifyPendingOrder(result.PendingOrder, Symbol.Ask + 20 * Symbol.PipSize ,null, null,
null);
}
private void PendingOrdersOnModified(PendingOrderModifiedEventArgs args)
{
    Print("Pending order with id {0} was modifed", args.PendingOrder.Id);
}
```

Cancelled

Summary

Occurs when pending order is cancelled

Syntax

```

```

```
public event Action Cancelled
```

Example 1

```
protected override void OnStart()
{
    PendingOrders.Cancelled += PendingOrdersOnCancelled;
    var result = PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask + 10 *
Symbol.PipSize);
    CancelPendingOrder(result.PendingOrder);
}
private void PendingOrdersOnCancelled(PendingOrderCancelledEventArgs args)
{
   Print("Pending order with id {0} was cancelled. Reason: {1}", args.PendingOrder.Id,
args.Reason);
}
```

Filled

Summary

Occurs when pending order is filled

Syntax

```
public event Action Filled
```

Example 1

```
protected override void OnStart()
{
    PendingOrders.Filled += PendingOrdersOnFilled;
    PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask);
}
private void PendingOrdersOnFilled(PendingOrderFilledEventArgs args)
{
    Print("Pending order with id {0} was filled, position id  is {1}", args.PendngOrder.Id,
args.Position.Id);
}
```

PendingOrderType

Summary

Represents the type (Limit or Stop) of pending order.

Syntax

```
public sealed enum PendingOrderType
```

Members

Name	Type	Summary
Limit	Field	A limit order is an order to buy or sell at a specific price or better.
Stop	Field	A stop order is an order to buy or sell once the price of the symbol reaches a specified price.
StopLimit	Field	A stop limit order is an order to buy or sell once the price of the symbol reaches specific price. Order has a parameter for maximum distance from that target price, where it can be executed.

Example 1

```
if(PendingOrders.Count > 0)
{
    PendingOrderType type = PendingOrders[0].OrderType;
}
```

Limit

Summary

A limit order is an order to buy or sell at a specific price or better.

Syntax

```
PendingOrderType.Limit
```

Example 1

```
foreach (var order in PendingOrders)
{
```

```
    if(order.OrderType == PendingOrderType.Limit)
        Print(order.Id);
}
```

Stop

Summary

A stop order is an order to buy or sell once the price of the symbol reaches a specified price.

Syntax

```
PendingOrderType.Stop
```

Example 1

```
foreach (var order in PendingOrders)
{
    if(order.OrderType == PendingOrderType.Stop)
        Print(order.Id);
}
```

StopLimit

Summary

A stop limit order is an order to buy or sell once the price of the symbol reaches specific price. Order has a parameter for maximum distance from that target price, where it can be executed.

Syntax

```
PendingOrderType.StopLimit
```

Example 1

```
foreach (var order in PendingOrders)
{
    if(order.OrderType == PendingOrderType.StopLimit)
```

```
        Print(order.Id);
}
```

PlotType

Summary

Plot type.

Syntax

```
public sealed enum PlotType
```

Members

Name	Type	Summary
DiscontinuousLine	Field	Plot Indicator result as a line with breaks where there are no values in the IndicatorDataSeries.
Histogram (3)	Field	Plot Indicator result as a histogram.
Line (2)	Field	Plot Indicator result as a line.
Points	Field	Plot Indicator result as a sequence of points.

Line

Summary

Plot Indicator result as a line.

Syntax

```
PlotType.Line
```

Example 1

```
[Output("Main", PlotType = PlotType.Line)]
public IndicatorDataSeries Result { get; set; }
```

Histogram

Summary

Plot Indicator result as a histogram.

Syntax

```
PlotType.Histogram
```

Example 1

```
[Output("Main", PlotType = PlotType.Histogram)]
public IndicatorDataSeries Result { get; set; }
```

Points

Summary

Plot Indicator result as a sequence of points.

Syntax

```
PlotType.Points
```

Example 1

```
[Output("Main", PlotType = PlotType.Points)]
public IndicatorDataSeries Result { get; set; }
```

DiscontinuousLine

Summary

Plot Indicator result as a line with breaks where there are no values in the IndicatorDataSeries.

Syntax

```
PlotType.DiscontinuousLine
```

Example 1

```
[Output("Main", PlotType = PlotType.DiscontinuousLine)]
public IndicatorDataSeries Result { get; set; }
```

Position

Summary

Taking or opening a position means buying or selling a trading pair.

Syntax

```
public interface Position
```

Members

Name	Type	Summary
Close (2)	Method	Shortcut for the Robot.ClosePosition method.
Comment (4)	Property	Comment can be used as a note for the order.
Commissions (2)	Property	Commission Amount of the request to trade one way (Buy/Sell) associated with this position.
EntryPrice (2)	Property	Entry price of the position.
EntryTime (2)	Property	Entry time of trade associated with the position. The Timezone used is set in the cBot attribute.
GrossProfit (2)	Property	Gross profit accrued by the order associated with the position.
HasTrailingStop (2)	Property	When HasTrailingStop set to true, the server updates the Stop Loss every time the position moves in your favor.
Id (2)	Property	The position's unique identifier.
Label (3)	Property	Label can be used to represent the order.
ModifyStopLossPips (2)	Method	Shortcut for the Robot.ModifyPosition method to change the Stop Loss pips
ModifyStopLossPrice	Method	Shortcut for Robot.ModifyPosition method to change the Stop Loss.

ModifyTakeProfitPips (2)	Method	Shortcut for the Robot.ModifyPosition method to change the Take Profit pips
ModifyTakeProfitPrice	Method	Shortcut for Robot.ModifyPosition method to change the Take Profit.
ModifyTrailingStop	Method	Shortcut for the Robot.ModifyPosition method to change the Trailing Stop.
ModifyVolume (2)	Method	Shortcut for the Robot.ModifyPosition method to change the VolumeInUnits.
NetProfit (3)	Property	The Net profit of the position.
Pips (2)	Property	Represents the winning or loosing pips of the position.
Quantity (3)	Property	Quantity of lots traded by the position.
Reverse	Method	Shortcut for the Robot.ReversePosition method to change the direction of the trade.
StopLoss (2)	Property	The Stop Loss level of the position.
StopLossTriggerMethod (2)	Property	Trigger method for the position's Stop Loss.
Swap (2)	Property	Swap is the overnight interest rate if any, accrued on the position.
SymbolCode (4)	Property	Symbol code of the position.
TakeProfit (2)	Property	The take profit level of the position.
TradeType (3)	Property	Trade type (Buy/Sell) of the position.
VolumeInUnits (4)	Property	The amount traded by the position.

Example 1

```
protected override void OnStart()
{
    foreach (var position in Positions)
    {
        Print("Position Label {0}", position.Label);
        Print("Position ID {0}", position.Id);
        Print("Profit {0}", position.GrossProfit);
        Print("Entry Price {0}", position.EntryPrice);
    }
}
```

SymbolCode

Summary

Symbol code of the position.

Syntax

```
public string SymbolCode{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.SymbolCode);
```

TradeType

Summary

Trade type (Buy/Sell) of the position.

Syntax

```
public TradeType TradeType{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.TradeType);
```

VolumeInUnits

Summary

The amount traded by the position.

Syntax

```
public double VolumeInUnits{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.VolumeInUnits);
```

Id

Summary

The position's unique identifier.

Syntax

```
public int Id{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.Id);
```

GrossProfit

Summary

Gross profit accrued by the order associated with the position.

Syntax

```
public double GrossProfit{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.GrossProfit);
```

EntryPrice

Summary

Entry price of the position.

Syntax

```
public double EntryPrice{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.EntryPrice);
```

StopLoss

Summary

The Stop Loss level of the position.

Syntax

```
public double? StopLoss{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.StopLoss);
```

TakeProfit

Summary

The take profit level of the position.

Syntax

```
public double? TakeProfit{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.TakeProfit);
```

NetProfit

Summary

The Net profit of the position.

Syntax

```
public double NetProfit{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.NetProfit);
```

Swap

Summary

Swap is the overnight interest rate if any, accrued on the position.

Syntax

```
public double Swap{ get; }
```

Example 1

```
Print(LastResult.Position.Swap);
```

Commissions

Summary

Commission Amount of the request to trade one way (Buy/Sell) associated with this position.

Syntax

```
public double Commissions{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.Commissions);
```

EntryTime

Summary

Entry time of trade associated with the position. The Timezone used is set in the cBot attribute.

Syntax

```
public DateTime EntryTime{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.EntryTime);
```

Pips

Summary

Represents the winning or loosing pips of the position.

Syntax

```
public double Pips{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10,10);
Print(LastResult.Position.Pips);
```

Label

Summary

Label can be used to represent the order.

Syntax

```
public string Label{ get; }
```

Example 1

```
var result = ExecuteMarketOrder(TradeType.Sell, Symbol, 10000, "myLabel");
if(result.IsSuccessful)
    Print("Position {0} is open", result.Position.Label);
```

Comment

Summary

Comment can be used as a note for the order.

Syntax

```
public string Comment{ get; }
```

Example 1

```
```

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 5000, "myLabel", 10, 10, 2, "this is a comment");
if(result.IsSuccessful)
    Print("Position is open: {0}", result.Position.Comment);
```

Quantity

Summary

Quantity of lots traded by the position.

Syntax

```
public double Quantity{ get; }
```

HasTrailingStop

Summary

When HasTrailingStop set to true, the server updates the Stop Loss every time the position moves in your favor.

Syntax

```
public bool HasTrailingStop{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10, 10, 2, "comment", true);
Print("Position was opened, has Trailing Stop = {0}", result.Position.HasTrailingStop);
```

StopLossTriggerMethod

Summary

Trigger method for the position's Stop Loss.

Syntax

```
public StopTriggerMethod? StopLossTriggerMethod{ get; }
```

ModifyStopLossPrice

Summary

Shortcut for Robot.ModifyPosition method to change the Stop Loss.

Syntax

```
public TradeResult ModifyStopLossPrice(double? stopLoss)
```

Parameters

Name	Description

ModifyTakeProfitPrice

Summary

Shortcut for Robot.ModifyPosition method to change the Take Profit.

Syntax

```
public TradeResult ModifyTakeProfitPrice(double? takeProfit)
```

Parameters

Name	Description

ModifyStopLossPips

Summary

Shortcut for the Robot.ModifyPosition method to change the Stop Loss pips

Syntax

```
public TradeResult ModifyStopLossPips(double? stopLossPips)
```

Parameters

Name	Description

ModifyTakeProfitPips

Summary

Shortcut for the Robot.ModifyPosition method to change the Take Profit pips

Syntax

```
public TradeResult ModifyTakeProfitPips(double? takeProfitPips)
```

Parameters

Name	Description

ModifyTrailingStop

Summary

Shortcut for the Robot.ModifyPosition method to change the Trailing Stop.

Syntax

```
public TradeResult ModifyTrailingStop(bool hasTrailingStop)
```

Parameters

Name	Description

ModifyVolume

Summary

Shortcut for the Robot.ModifyPosition method to change the VolumeInUnits.

Syntax

```
public TradeResult ModifyVolume(double volume)
```

Parameters

Name	Description

Reverse

Summary

Shortcut for the Robot.ReversePosition method to change the direction of the trade.

Syntax

```
public TradeResult Reverse()
```

```
public TradeResult Reverse(double volume)
```

Reverse

Summary

Shortcut for the Robot.ReversePosition method to change the direction of trade and the volume.

Syntax

```
public TradeResult Reverse()
```

```
public TradeResult Reverse(double volume)
```

Parameters

Name	Description

Close

Summary

Shortcut for the Robot.ClosePosition method.

Syntax

```
public TradeResult Close()
```

PositionClosedEventArgs

Summary

Provides data for the position closing event.

Syntax

```
public class PositionClosedEventArgs : Object
```

Members

Name	Type	Summary
Position (2)	Property	Gets the position being closed.
PositionClosedEventArgs	Method	
Reason (2)	Property	Gets the reason of the position being closed.

Example 1

```
protected override void OnStart()
{
    Positions.Closed += PositionsClosed;
}
private void PositionsOnClosed(PositionClosedEventArgs args)
{
    var position = args.Position;
```

```
    Print("Position closed with {0} profit", position.GrossProfit);
}
```

Position

Summary

Gets the position being closed.

Syntax

```
public Position Position{ get; }
```

Example 1

```
protected override void OnStart()
{
    ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel");
    Positions.Closed += PositionsClosed;
}
private void PositionsClosed(PositionClosedEventArgs args)
{
    var position = args.Position;
    if(position.Label == "myLabel")
    {
        var tradeType = position.TradeType;
        var symbol = MarketData.GetSymbol(position.SymbolCode);
        var volume = position.Volume;
        var label = position.Label;

        if(position.GrossProfit > 0)
            ExecuteMarketOrder(tradeType, symbol, volume, label);
        else
        {
            var oppositeTrade = tradeType == TradeType.Buy
                        ? TradeType.Sell
                        : TradeType.Buy;
            ExecuteMarketOrder(oppositeTrade, symbol, volume, label);
        }
    }
}
```

Reason

Summary

Gets the reason of the position being closed.

Syntax

```
public PositionCloseReason Reason{ get; }
```

PositionClosedEventArgs

Syntax

```
protected PositionClosedEventArgs PositionClosedEventArgs(Position position,
PositionCloseReason reason)
```

Parameters

Name	Description

PositionCloseReason

Summary

Reason for position closing

Syntax

```
public sealed enum PositionCloseReason
```

Members

Name	Type	Summary
Closed	Field	Positions was closed by trader
StopLoss (3)	Field	Position was closed by Stop Loss
StopOut	Field	Position was closed because Stop Out level reached

TakeProfit (3)	Field	Position was closed by Take Profit

Closed

Summary

Positions was closed by trader

Syntax

```
PositionCloseReason.Closed
```

StopLoss

Summary

Position was closed by Stop Loss

Syntax

```
PositionCloseReason.StopLoss
```

TakeProfit

Summary

Position was closed by Take Profit

Syntax

```
PositionCloseReason.TakeProfit
```

StopOut

Summary

Position was closed because Stop Out level reached

Syntax

```
PositionCloseReason.StopOut
```

PositionModifiedEventArgs

Summary

Provides data for the position modification event.

Syntax

```
public class PositionModifiedEventArgs : Object
```

Members

Name	Type	Summary
Position (3)	Property	Gets or sets the modified position.

Position

Summary

Gets or sets the modified position.

Syntax

```
public Position Position{ get; }
```

PositionOpenedEventArgs

Summary

Provides data for the position opening event.

Syntax

```
public class PositionOpenedEventArgs : Object
```

Members

Name	Type	Summary
Position (4)	Property	Gets or sets the position being opened.

Example 1

```
public class SampleRobot : Robot
{
    protected override void OnStart()
    {
        Positions.Opened += Positions_Opened;
        ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel");
    }
    private void Positions_Opened(PositionOpenedEventArgs args)
    {
        var position = args.Position;
        if(position.Label == "myLabel")
            Print("Position opened by SampleRobot");
    }
    //...
}
```

Position

Summary

Gets or sets the position being opened.

Syntax

```
public Position Position{ get; }
```

Example 1

```

```

```
private void PositionsOnOpened(PositionOpenedEventArgs args)
{
    var position = args.Position;
    Print("Position opened at {0}", position.EntryPrice);
}
```

Positions

Summary

Provides access to methods of the positions collection.

Syntax

```
public interface Positions : IEnumerable
```

Members

Name	Type	Summary
Closed (2)	Event	Occurs each time a position is closed.
Count (5)	Property	The total number of open positions.
Find	Method	Find a position by its label.
FindAll (2)	Method	Find all positions with this label.
Modified (2)	Event	
Opened	Event	Occurs each time a position is opened.
this[int index] (6)	Property	Finds a position by index.

Example 1

```
int totalPositions = Positions.Count;
```

Example 2

```
Position position =  Positions.Find("myLabel", Symbol, TradeType.Buy);
```

Example 3

```

```

```
Position[] positions = Positions.FindAll("myLabel", Symbol, TradeType.Buy);
```

Example 4

```
Positions.Opened += PositionsOnOpened;
```

this[int index]

Summary

Finds a position by index.

Syntax

```
public Position this[int index]{ get; }
```

Parameters

Name	Description

Example 1

```
var position = Positions[0];
```

Count

Summary

The total number of open positions.

Syntax

```
public int Count{ get; }
```

Example 1

```

```

```
int totalPositions = Positions.Count;
```

Find

Summary

Find a position by its label.

Syntax

```
public Position Find(string label)
```

```
public Position Find(string label, Symbol symbol)
```

```
public Position Find(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel");
```

Find

Summary

Find a position by its label and symbol.

Syntax

```
public Position Find(string label)
```

```
public Position Find(string label, Symbol symbol)
```

```
public Position Find(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol);
```

Find

Summary

Find a position by its label, symbol and trade type

Syntax

```
public Position Find(string label)
```

```
public Position Find(string label, Symbol symbol)
```

```
public Position Find(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
```

FindAll

Summary

Find all positions with this label.

Syntax

```
public Position[] FindAll(string label)
```

```
public Position[] FindAll(string label, Symbol symbol)
```

```
public Position[] FindAll(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

Example 1

```
var positions = Positions.FindAll("myLabel");
foreach (var position in positions)
{
    double? newStopLoss = position.StopLoss ?? 10;
    ModifyPosition(position, newStopLoss, position.TakeProfit);
}
```

FindAll

Summary

Find all positions with this label and symbol.

Syntax

```
public Position[] FindAll(string label)
```

```
public Position[] FindAll(string label, Symbol symbol)
```

```
```

```
public Position[] FindAll(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

Example 1

```
var positions = Positions.FindAll("myLabel", Symbol);
foreach (var position in positions)
{
    double? newStopLoss = position.StopLoss ?? 10;
    ModifyPosition(position, newStopLoss, position.TakeProfit);
}
```

FindAll

Summary

Finds all the positions of this label, symbol and trade type.

Syntax

```
public Position[] FindAll(string label)
```

```
public Position[] FindAll(string label, Symbol symbol)
```

```
public Position[] FindAll(string label, Symbol symbol, TradeType tradeType)
```

Parameters

Name	Description

Example 1

```
var positions = Positions.FindAll("myLabel", Symbol, TradeType.Buy);
foreach (var position in positions)
{
    double? newStopLoss = position.StopLoss ?? 10;
```

```
    ModifyPosition(position, newStopLoss, position.TakeProfit);
}
```

Closed

Summary

Occurs each time a position is closed.

Syntax

```
public event Action Closed
```

Example 1

```
protected override void OnStart()
 {
     Positions.Closed += PositionsOnClosed;
 }
 private void PositionsOnClosed(PositionClosedEventArgs args)
 {
     var position = args.Position;
     Print("Position closed with {0} profit", position.GrossProfit);
 }
```

Opened

Summary

Occurs each time a position is opened.

Syntax

```
public event Action Opened
```

Example 1

```
protected override void OnStart()
```

```
{
    Positions.Opened += PositionsOnOpened;
}
private void PositionsOnOpened(PositionOpenedEventArgs args)
{
    Print("Position opened {0}", args.Position.Label);
}
```

Modified

Syntax

```
public event Action Modified
```

Robot

Summary

Base class for all cBots.

Remarks

Provides a convenient framework for creating cBots including methods to create, modify, cancel orders and close positions, methods trigered by each tick and each bar, access to built-in Indicators and more.

Syntax

```
public class Robot : Algo
```

Members

Name	Type	Summary
Account (2)	Property	Contains all Account information
CancelPendingOrder	Method	Cancel a Pending Order
CancelPendingOrderAsync	Method	Cancel a Pending Order in asynchronous execution mode
ClosePosition	Method	Close a position
ClosePositionAsync	Method	Close a position in asynchronous execution mode
ExecuteMarketOrder	Method	Execute a Market Order

ExecuteMarketOrderAsync	Method	Execute a market order in asynchronous execution mode
GetFitness	Method	Override this method to provide custom fitness value for Optimization
LastResult	Property	The latest trade result
ModifyPendingOrder	Method	Modify a Pending Order
ModifyPendingOrderAsync	Method	Modify a Pending Order in asynchronous execution mode
ModifyPosition	Method	Modify the volume of a position
ModifyPositionAsync	Method	Modify Position in asynchronous execution mode
OnBar	Method	Called on each incoming Bar.
OnError	Method	Called if there is an error executing a trade operation.
OnStart	Method	Called when cBot is being started. Override this method to initialize cBot, create nested indicators, etc.
OnStop	Method	Called when cBot is stopped.
OnTick	Method	Called on each incoming market tick.
PlaceLimitOrder	Method	Place a Limit Order
PlaceLimitOrderAsync	Method	Place limit order in asynchronous execution mode
PlaceStopLimitOrder	Method	Place a Stop Limit Order
PlaceStopLimitOrderAsync	Method	Place Stop Limit order in asynchronous execution mode
PlaceStopOrder	Method	Place a stop order
PlaceStopOrderAsync	Method	Place stop order in asynchronous execution mode
ReversePosition	Method	Modify the direction of trade at position
ReversePositionAsync	Method	Modify Position in asynchronous execution mode
Robot	Method	Robot class constructor
Stop (2)	Method	Stops the cBot. cBot will be completely stopped and will not send/receive any signals.
ToString (3)	Method	Returns the cBot class name

Example 1

```
namespace cAlgo.Robots
{
[Robot]
public class myCBot : Robot
{
    protected override void OnStart()
    {
        //This method is called when the cBot is being started, once.
    }

    protected override void OnBar()
```

```
    {
        // Called on each incoming Bar.
    }
    protected override void OnTick()
    {
        // Called on each incoming tick.
    }

    protected override void OnError(Error error)
    {
        Print("There has been an Error");
    }
    protected override void OnStop()
    {
        //This method is called when the cBot is being stoped.
    }
```

Account

Summary

Contains all Account information

Syntax

```
public IAccount Account{ get; }
```

Example 1

```
double balance = Account.Balance;
string currency = Account.Currency;
double equity = Account.Equity;
double freemargin = Account.FreeMargin;
double margin = Account.Margin;
double? marginlevel = Account.MarginLevel;
int leverage = Account.Leverage;
```

LastResult

Summary

The latest trade result

Syntax

```
public TradeResult LastResult{ get; }
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 20000, null, 10, null);
if(LastResult.IsSuccessful)
    Print(LastResult.Position.StopLoss);
```

Robot

Summary

Robot class constructor

Syntax

```
public Robot Robot()
```

OnStart

Summary

Called when cBot is being started. Override this method to initialize cBot, create nested indicators, etc.

Syntax

```
protected virtual void OnStart()
```

Example 1

```
protected override void OnStart()
{
    //This method is invoked when the cBot is started.
}
```

OnStop

Summary

Called when cBot is stopped.

Syntax

```
protected virtual void OnStop()
```

Example 1

```
protected override void OnStop()
{
    //This method is called when the cBot is stopped
}
```

OnTick

Summary

Called on each incoming market tick.

Syntax

```
protected virtual void OnTick()
```

Example 1

```
protected override void OnTick()
{
    // Place cBot's Logic here.
}
```

OnBar

Summary

Called on each incoming Bar.

Syntax

```
protected virtual void OnBar()
```

Example 1

```
protected override void OnBar()
{
    //Place cBot's Logic here.
}
```

OnError

Summary

Called if there is an error executing a trade operation.

Syntax

```
protected virtual void OnError(Error error)
```

Parameters

Name	Description

Example 1

```
protected override void OnError(Error error)
{
    Print("There has been an Error");
}
```

Stop

Summary

Stops the cBot. cBot will be completely stopped and will not send/receive any signals.

Syntax

```
public void Stop()
```

Example 1

```
// Will stop the cBot if the balance of the account goes under 1000
if(Account.Balance < 1000)
{
    Stop();
}
```

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label)
```

```

```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips)
```

```
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
```

```
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips)
```

```

```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000);
```

Example 2

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000, "Robot1");
```

Example 3

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10, 10);
```

Example 4

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000, "Robot1", 10, 10, 2);
```

Example 5

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 5000, "myRobot", StopLoss, TakeProfit, Slippage,
"this is a comment");
```

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000);
```

Example 2

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000, "Robot1");
```

Example 3

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10, 10);
```

Example 4

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000, "Robot1", 10, 10, 2);
```

Example 5

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 5000, "myRobot", StopLoss, TakeProfit, Slippage,
"this is a comment");
```

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000);
```

Example 2

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000, "Robot1");
```

Example 3

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10, 10);
```

Example 4

```
```

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000, "Robot1", 10, 10, 2);
```

Example 5

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 5000, "myRobot", StopLoss, TakeProfit, Slippage,
"this is a comment");
```

Example 6

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 5000, "myRobot", StopLoss, TakeProfit, Slippage,
"this is a comment", HasTrailingStop);
```

ExecuteMarketOrder

Summary

Execute a Market Order

Syntax

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string
label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume,
string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop)
```

```
public TradeResult ExecuteMarketOrder(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000);
```

Example 2

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000, "Robot1");
```

Example 3

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "myLabel", 10, 10);
```

Example 4

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000, "Robot1", 10, 10, 2);
```

Example 5

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 5000, "myRobot", StopLoss, TakeProfit, Slippage,
"this is a comment");
```

Example 6

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 5000, "myRobot", StopLoss, TakeProfit, Slippage,
"this is a comment", HasTrailingStop);
```

Example 7

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 5000, "myRobot", StopLoss, TakeProfit, Slippage,
"this is a comment", HasTrailingStop, StopTriggerMethod.Trade);
```

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
```

```
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label)
```

```
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
```

```
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
PlaceLimitOrder(TradeType.Buy, Symbol, 100000,
                Symbol.Bid - 2*Symbol.PipSize);
```

Example 2

```
PlaceLimitOrder(TradeType.Buy, Symbol, 200000,
            Symbol.Bid - 2*Symbol.PipSize, "myLabel");
```

Example 3

```
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                Symbol.Bid - 5*Symbol.PipSize, "112", 10, 10);
```

Example 4

```
double targetPrice = Symbol.Bid - 5*Symbol.PipSize;
DateTime expiry = DateTime.Now.AddMinutes(30);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                targetPrice, "112", 10, 10, expiry);
```

Example 5

```
double targetPrice = Symbol.Bid - 5*Symbol.PipSize;
DateTime expiry = DateTime.Now.AddMinutes(30);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                targetPrice, "112", 10, 10, expiry, "first order");
```

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
PlaceLimitOrder(TradeType.Buy, Symbol, 100000,
                Symbol.Bid - 2*Symbol.PipSize);
```

Example 2

```
PlaceLimitOrder(TradeType.Buy, Symbol, 200000,
            Symbol.Bid - 2*Symbol.PipSize, "myLabel");
```

Example 3

```
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                Symbol.Bid - 5*Symbol.PipSize, "112", 10, 10);
```

Example 4

```
double targetPrice = Symbol.Bid - 5*Symbol.PipSize;
DateTime expiry = DateTime.Now.AddMinutes(30);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                targetPrice, "112", 10, 10, expiry);
```

Example 5

```
double targetPrice = Symbol.Bid - 5*Symbol.PipSize;
DateTime expiry = DateTime.Now.AddMinutes(30);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                targetPrice, "112", 10, 10, expiry, "first order");
```

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
```

```
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
PlaceLimitOrder(TradeType.Buy, Symbol, 100000,
                Symbol.Bid - 2*Symbol.PipSize);
```

Example 2

```
PlaceLimitOrder(TradeType.Buy, Symbol, 200000,
            Symbol.Bid - 2*Symbol.PipSize, "myLabel");
```

Example 3

```
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                Symbol.Bid - 5*Symbol.PipSize, "112", 10, 10);
```

Example 4

```
```

```
double targetPrice = Symbol.Bid - 5*Symbol.PipSize;
DateTime expiry = DateTime.Now.AddMinutes(30);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                targetPrice, "112", 10, 10, expiry);
```

Example 5

```
double targetPrice = Symbol.Bid - 5*Symbol.PipSize;
DateTime expiry = DateTime.Now.AddMinutes(30);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                targetPrice, "112", 10, 10, expiry, "first order");
```

Example 6

```
double targetPrice = Symbol.Bid - 5*Symbol.PipSize;
DateTime expiry = DateTime.Now.AddMinutes(30);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                targetPrice, "112", 10, 10, expiry, "first order", HasTrailingStop);
```

PlaceLimitOrder

Summary

Place a Limit Order

Syntax

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceLimitOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
PlaceLimitOrder(TradeType.Buy, Symbol, 100000,
                Symbol.Bid - 2*Symbol.PipSize);
```

Example 2

```
PlaceLimitOrder(TradeType.Buy, Symbol, 200000,
            Symbol.Bid - 2*Symbol.PipSize, "myLabel");
```

Example 3

```
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                Symbol.Bid - 5*Symbol.PipSize, "112", 10, 10);
```

Example 4

```
double targetPrice = Symbol.Bid - 5*Symbol.PipSize;
DateTime expiry = DateTime.Now.AddMinutes(30);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                targetPrice, "112", 10, 10, expiry);
```

Example 5

```
double targetPrice = Symbol.Bid - 5*Symbol.PipSize;
DateTime expiry = DateTime.Now.AddMinutes(30);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                targetPrice, "112", 10, 10, expiry, "first order");
```

Example 6

```
double targetPrice = Symbol.Bid - 5*Symbol.PipSize;
DateTime expiry = DateTime.Now.AddMinutes(30);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                targetPrice, "112", 10, 10, expiry, "first order", HasTrailingStop);
```

Example 7

```
double targetPrice = Symbol.Bid - 5*Symbol.PipSize;
DateTime expiry = DateTime.Now.AddMinutes(30);
PlaceLimitOrder(TradeType.Buy, Symbol, 10000,
                targetPrice, "112", 10, 10, expiry, "first order", HasTrailingStop,
StopTriggerMethod.Trade);
```

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
```

```
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
```

```
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
```

```
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopOrder

Summary

Place a stop order

Syntax

```
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
```

```
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

Example 1

```
PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask);
```

Example 2

```
PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask,
                "myStopOrder");
```

Example 3

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20);
```

Example 4

```
DateTime expiration = Server.Time.AddHours(1);
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, expiration);
```

Example 5

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, null, "my comment");
```

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

Example 1

```
PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask);
```

Example 2

```
PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask,
                "myStopOrder");
```

Example 3

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20);
```

Example 4

```
DateTime expiration = Server.Time.AddHours(1);
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, expiration);
```

Example 5

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, null, "my comment");
```

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

Example 1

```
PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask);
```

Example 2

```
PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask,
                "myStopOrder");
```

Example 3

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20);
```

Example 4

```
DateTime expiration = Server.Time.AddHours(1);
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, expiration);
```

Example 5

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, null, "my comment");
```

Example 6

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, null, "my comment", HasTrailingStop);
```

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

Example 1

```
PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask);
```

Example 2

```
PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask,
                "myStopOrder");
```

Example 3

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20);
```

Example 4

```
DateTime expiration = Server.Time.AddHours(1);
```

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, expiration);
```

Example 5

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, null, "my comment");
```

Example 6

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, null, "my comment", HasTrailingStop);
```

Example 7

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, null, "my comment", HasTrailingStop,
StopTriggerMethod.Trade);
```

PlaceStopOrder

Summary

Place a stop order

Syntax

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
```

```
targetPrice, string label)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, long volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult PlaceStopOrder(TradeType tradeType, Symbol symbol, double volume, double
targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

Example 1

```
PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask);
```

Example 2

```
PlaceStopOrder(TradeType.Buy, Symbol, 10000, Symbol.Ask,
                "myStopOrder");
```

Example 3

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20);
```

Example 4

```
DateTime expiration = Server.Time.AddHours(1);
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, expiration);
```

Example 5

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, null, "my comment");
```

Example 6

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, null, "my comment", HasTrailingStop);
```

Example 7

```
PlaceStopOrder(TradeType.Sell, Symbol, 20000, Symbol.Ask,
                "myStopOrder", 20, 20, null, "my comment", HasTrailingStop,
StopTriggerMethod.Trade);
```

CancelPendingOrder

Summary

Cancel a Pending Order

Syntax

```
public TradeResult CancelPendingOrder(PendingOrder pendingOrder)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    CancelPendingOrder(order);
}
```

ModifyPendingOrder

Summary

Modify a Pending Order

Syntax

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
```

```
stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice);
}
```

ModifyPendingOrder

Summary

Modify a Pending Order

Syntax

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips);

}
```

ModifyPendingOrder

Summary

Modify a Pending Order

Syntax

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime);
}
```

ModifyPendingOrder

Summary

Modify a Pending Order

Syntax

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime, 5);
}
```

ModifyPendingOrder

Summary

Modify a Pending Order

Syntax

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
                           order.ExpirationTime, 5);
}
```

ModifyPendingOrder

Summary

Modify a Pending Order

Syntax

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
```

```
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime);
}
```

Example 2

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime, hasTrailingStop);
}
```

ModifyPendingOrder

Summary

Modify a Pending Order

Syntax

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips)
```

Parameters

Name	Description

Example 1

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
                           order.ExpirationTime, 10000, hasTrailingStop,
StopTriggerMethod.Trade);
}
```

ModifyPendingOrder

Summary

Modify a Pending Order

Syntax

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
                           order.ExpirationTime);
}
```

Example 2

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
if (order.StopLossPips == null)
    ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
```

```
                        order.ExpirationTime, 5, hasTrailingStop);
}
```

Example 3

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
                           order.ExpirationTime, 5, hasTrailingStop, StopTriggerMethod.Trade,
StopTriggerMethod.Trade);
}
```

ModifyPendingOrder

Summary

Modify a Pending Order

Syntax

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
```

```
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod)
```

```
public TradeResult ModifyPendingOrder(PendingOrder pendingOrder, double targetPrice, double?
stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
                           order.ExpirationTime);
}
```

Example 2

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
if (order.StopLossPips == null)
    ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
                       order.ExpirationTime, 5, hasTrailingStop);
}
```

Example 3

```

```

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrder(order, order.TargetPrice, 10, order.TakeProfitPips,
                           order.ExpirationTime, 5, hasTrailingStop, StopTriggerMethod.Trade,
StopTriggerMethod.Trade, 2);
}
```

ReversePosition

Summary

Modify the direction of trade at position

Syntax

```
public TradeResult ReversePosition(Position position)
```

```
public TradeResult ReversePosition(Position position, double volume)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null )
{
    ReversePosition(position);
}
```

ModifyPosition

Summary

Modify the volume of a position

Syntax

```
public TradeResult ModifyPosition(Position position, double volume)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit,
bool hasTrailingStop)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit,
bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null )
{
    ModifyPosition(position, 20000);
}
```

ReversePosition

Summary

Modify the direction of trade and volume of a position

Syntax

```
public TradeResult ReversePosition(Position position)
```

```
public TradeResult ReversePosition(Position position, double volume)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null )
{
    ReversePosition(position, 20000);
}
```

ModifyPosition

Summary

Modify the protection of a position

Syntax

```
public TradeResult ModifyPosition(Position position, double volume)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit,
bool hasTrailingStop)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit,
bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null )
{
    double? stopLoss = Symbol.Ask- 10*Symbol.PipSize;
```

```
    double? takeProfit = Symbol.Ask + 10 * Symbol.PipSize;
    ModifyPosition(position, stopLoss,  takeProfit);
}
```

ModifyPosition

Summary

Modify the protection of a position

Syntax

```
public TradeResult ModifyPosition(Position position, double volume)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit,
bool hasTrailingStop)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit,
bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null )
{
    double? stopLoss = Symbol.Ask- 10*Symbol.PipSize;
    double? takeProfit = Symbol.Ask + 10 * Symbol.PipSize;
    ModifyPosition(position, stopLoss,  takeProfit);
}
```

Example 2

```

```

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null )
{
    double? stopLoss = Symbol.Ask- 10*Symbol.PipSize;
    double? takeProfit = Symbol.Ask + 10 * Symbol.PipSize;
    bool hasTrailingStop = true;
    ModifyPosition(position, stopLoss,  takeProfit, hasTrailingStop);
    Print("Position was modified, has Trailing Stop = {0}", result.Position.HasTrailingStop);
}
```

ModifyPosition

Summary

Modify the protection of a position

Syntax

```
public TradeResult ModifyPosition(Position position, double volume)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit,
bool hasTrailingStop)
```

```
public TradeResult ModifyPosition(Position position, double? stopLoss, double? takeProfit,
bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null )
{
    double? stopLoss = Symbol.Ask- 10*Symbol.PipSize;
    double? takeProfit = Symbol.Ask + 10 * Symbol.PipSize;
```

```
    ModifyPosition(position, stopLoss,  takeProfit);
}
```

Example 2

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null )
{
    double? stopLoss = Symbol.Ask- 10*Symbol.PipSize;
    double? takeProfit = Symbol.Ask + 10 * Symbol.PipSize;
    bool hasTrailingStop = true;
    ModifyPosition(position, stopLoss,  takeProfit, hasTrailingStop);
    Print("Position was modified, has Trailing Stop = {0}", result.Position.HasTrailingStop);
}
```

Example 3

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null )
{
    double? stopLoss = Symbol.Ask- 10*Symbol.PipSize;
    double? takeProfit = Symbol.Ask + 10 * Symbol.PipSize;
    bool hasTrailingStop = true;
    ModifyPosition(position, stopLoss,  takeProfit, hasTrailingStop,
StopTriggerMethod.Opposite);
    Print("Position was modified, stop loss trigger method = {0}",
result.Position.StopLossTriggerMethod);
}
```

ClosePosition

Summary

Close a position

Syntax

```
public TradeResult ClosePosition(Position position)
```

```
public TradeResult ClosePosition(Position position, long volume)
```

```
public TradeResult ClosePosition(Position position, double volume)
```

Parameters

Name	Description

ClosePosition

Summary

Close a position

Syntax

```
public TradeResult ClosePosition(Position position)
```

```
public TradeResult ClosePosition(Position position, long volume)
```

```
public TradeResult ClosePosition(Position position, double volume)
```

Parameters

Name	Description

Example 1

```
ClosePosition(position);
```

Example 2

```
if (position.Volume >= 20000)
    ClosePosition(position, 10000);
```

ClosePosition

Summary

Close a position

Syntax

```
public TradeResult ClosePosition(Position position)
```

```
public TradeResult ClosePosition(Position position, long volume)
```

```
public TradeResult ClosePosition(Position position, double volume)
```

Parameters

Name	Description

Example 1

```
ClosePosition(position);
```

Example 2

```
if (position.Volume >= 20000)
    ClosePosition(position, 10000);
```

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action
callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
```

```
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action
callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action
callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
```

```
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, [optional] Action callback)
```

```
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
```

```
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, [optional] Action callback)
```

```
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
```

```
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, [optional] Action callback)
```

```
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000);
```

Example 2

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel");
```

Example 3

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20);
```

Example 4

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2);
```

Example 5

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2, "order comment");
```

Example 6

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2,
                        "order comment", OnOpened);
```

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
```

```
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

Example 1

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000);
```

Example 2

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel");
```

Example 3

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20);
```

Example 4

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2);
```

Example 5

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2, "order comment");
```

Example 6

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2,
                        "order comment", OnOpened);
```

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
```

```
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000);
```

Example 2

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel");
```

Example 3

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20);
```

Example 4

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2);
```

Example 5

```
```

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2, "order comment");
```

Example 6

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2,
                        "order comment", OnOpened);
```

Example 7

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2,
                        "order comment", HasTrailingStop, OnOpened);
```

ExecuteMarketOrderAsync

Summary

Execute a market order in asynchronous execution mode

Syntax

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
```

```
volume, string label, double? stopLossPips, double? takeProfitPips, [optional] Action
callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
[optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, long volume,
string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips, string
comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ExecuteMarketOrderAsync(TradeType tradeType, Symbol symbol, double
volume, string label, double? stopLossPips, double? takeProfitPips, double? marketRangePips,
string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

Example 1

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000);
```

Example 2

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel");
```

Example 3

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20);
```

Example 4

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2);
```

Example 5

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2, "order comment");
```

Example 6

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2,
                        "order comment", OnOpened);
```

Example 7

```
ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000, "myLabel", 10, 20, 2,
                        "order comment", HasTrailingStop, OnOpened);
```

Example 8

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 5000, "myRobot", StopLoss, TakeProfit, Slippage,
                   "this is a comment", HasTrailingStop, StopTriggerMethod.Trade);
```

ClosePositionAsync

Summary

Close a position in asynchronous execution mode

Syntax

```
public TradeOperation ClosePositionAsync(Position position, [optional] Action callback)
```

```
public TradeOperation ClosePositionAsync(Position position, long volume, [optional] Action
callback)
```

```
public TradeOperation ClosePositionAsync(Position position, double volume, [optional] Action
callback)
```

Parameters

Name	Description

ClosePositionAsync

Summary

Close a position in asynchronous execution mode

Syntax

```
public TradeOperation ClosePositionAsync(Position position, [optional] Action callback)
```

```
public TradeOperation ClosePositionAsync(Position position, long volume, [optional] Action
callback)
```

```
public TradeOperation ClosePositionAsync(Position position, double volume, [optional] Action
callback)
```

Parameters

Name	Description

Example 1

```
ClosePositionAsync(position);
```

Example 2

```
```

```
if (position.Volume >= 20000)
    ClosePositionAsync(position, 10000);
```

ClosePositionAsync

Summary

Close a position in asynchronous execution mode

Syntax

```
public TradeOperation ClosePositionAsync(Position position, [optional] Action callback)
```

```
public TradeOperation ClosePositionAsync(Position position, long volume, [optional] Action
callback)
```

```
public TradeOperation ClosePositionAsync(Position position, double volume, [optional] Action
callback)
```

Parameters

Name	Description

Example 1

```
ClosePositionAsync(position);
```

Example 2

```
if (position.Volume >= 20000)
    ClosePositionAsync(position, 10000);
```

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
```

```
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
```

```
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

Example 1

```
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel");
```

Example 2

```
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10,10);
```

Example 3

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order
comment");
```

Example 4

```
protected override void OnStart()
{
    PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000, Symbol.Bid, LimitOrderOnPlaced);
}
```

```
private void LimitOrderOnPlaced(TradeResult tradeResult)
{
    Print("Limit order placed {0}", tradeResult.PendingOrder.Label);
}
```

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

Example 1

```
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel");
```

Example 2

```
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10,10);
```

Example 3

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
```

```
                    Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order
comment");
```

Example 4

```
protected override void OnStart()
{
    PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000, Symbol.Bid, LimitOrderOnPlaced);
}
private void LimitOrderOnPlaced(TradeResult tradeResult)
{
    Print("Limit order placed {0}", tradeResult.PendingOrder.Label);
}
```

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

Example 1

```
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel");
```

Example 2

```
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10,10);
```

Example 3

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order
comment");
```

Example 4

```
protected override void OnStart()
{
    PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000, Symbol.Bid, LimitOrderOnPlaced);
}
private void LimitOrderOnPlaced(TradeResult tradeResult)
{
    Print("Limit order placed {0}", tradeResult.PendingOrder.Label);
}
```

Example 5

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order
comment", HasTrailingStop);
```

PlaceLimitOrderAsync

Summary

Place limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

Parameters

Name	Description

Example 1

```
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel");
```

Example 2

```
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10,10);
```

Example 3

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order
comment");
```

Example 4

```
protected override void OnStart()
{
    PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000, Symbol.Bid, LimitOrderOnPlaced);
}
private void LimitOrderOnPlaced(TradeResult tradeResult)
{
    Print("Limit order placed {0}", tradeResult.PendingOrder.Label);
}
```

Example 5

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order
comment", HasTrailingStop);
```

Example 6

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceLimitOrderAsync(TradeType.Buy, Symbol, 10000,
                     Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order
comment", HasTrailingStop, StopTriggerMethod.Trade);
```

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```

```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
```

```
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
```

```
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
```

```
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
```

```
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
```

```
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000, Symbol.Bid - 5* Symbol.PipSize);
```

Example 2

```
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000, Symbol.Bid - 5* Symbol.PipSize, "myLabel",
10, 10);
```

Example 3

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000,
            Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order comment");
```

Example 4

```
protected override void OnStart()
{
    PlaceStopOrderAsync(TradeType.Buy, Symbol, 20000, Symbol.Ask, StopOrderOnPlaced);
}
private void StopOrderOnPlaced(TradeResult tradeResult)
{
    Print("Stop order placed {0}", tradeResult.PendingOrder.Label);
}
```

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
```

```
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000, Symbol.Bid - 5* Symbol.PipSize);
```

Example 2

```
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000, Symbol.Bid - 5* Symbol.PipSize, "myLabel",
10, 10);
```

Example 3

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000,
            Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order comment");
```

Example 4

```
protected override void OnStart()
{
    PlaceStopOrderAsync(TradeType.Buy, Symbol, 20000, Symbol.Ask, StopOrderOnPlaced);
}
private void StopOrderOnPlaced(TradeResult tradeResult)
{
    Print("Stop order placed {0}", tradeResult.PendingOrder.Label);
}
```

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
```

```
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000, Symbol.Bid - 5* Symbol.PipSize);
```

Example 2

```
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000, Symbol.Bid - 5* Symbol.PipSize, "myLabel",
10, 10);
```

Example 3

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000,
            Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order comment");
```

Example 4

```
protected override void OnStart()
{
```

```
    PlaceStopOrderAsync(TradeType.Buy, Symbol, 20000, Symbol.Ask, StopOrderOnPlaced);
}
private void StopOrderOnPlaced(TradeResult tradeResult)
{
    Print("Stop order placed {0}", tradeResult.PendingOrder.Label);
}
```

Example 5

```
protected override void OnStart()
{
    bool hasTrailingStop = true;
    DateTime? expiry = DateTime.Now.AddHours(1);
    PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000,
                        Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order
comment", hasTrailingStop);
}
private void StopOrderOnPlaced(TradeResult tradeResult)
{
    Print("Stop order placed with HasTrailingStop: {0}",
tradeResult.PendingOrder.HasTrailingStop);
}
```

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
```

```
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
```

```
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000, Symbol.Bid - 5* Symbol.PipSize);
```

Example 2

```
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000, Symbol.Bid - 5* Symbol.PipSize, "myLabel",
10, 10);
```

Example 3

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000,
            Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order comment");
```

Example 4

```
protected override void OnStart()
{
    PlaceStopOrderAsync(TradeType.Buy, Symbol, 20000, Symbol.Ask, StopOrderOnPlaced);
}
private void StopOrderOnPlaced(TradeResult tradeResult)
{
    Print("Stop order placed {0}", tradeResult.PendingOrder.Label);
}
```

Example 5

```
protected override void OnStart()
{
    bool hasTrailingStop = true;
    DateTime? expiry = DateTime.Now.AddHours(1);
    PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000,
                        Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order
comment", hasTrailingStop);
}
```

```
private void StopOrderOnPlaced(TradeResult tradeResult)
{
    Print("Stop order placed with HasTrailingStop: {0}",
tradeResult.PendingOrder.HasTrailingStop);
}
```

Example 6

```
protected override void OnStart()
{
    bool hasTrailingStop = true;
    DateTime? expiry = DateTime.Now.AddHours(1);
    PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000,
                        Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry,
                        "order comment", hasTrailingStop, StopTriggerMethod.Trade);
}
private void StopOrderOnPlaced(TradeResult tradeResult)
{
    Print("Stop order placed with stop trigger method: {0}",
tradeResult.PendingOrder.StopTriggerMethod);
}
```

PlaceStopOrderAsync

Summary

Place stop order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, [optional]
Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, long volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
[optional] Action callback)
```

```
public TradeOperation PlaceStopOrderAsync(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, string label, double? stopLossPips, double? takeProfitPips, DateTime?
expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod,
StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000, Symbol.Bid - 5* Symbol.PipSize);
```

Example 2

```
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000, Symbol.Bid - 5* Symbol.PipSize, "myLabel",
10, 10);
```

Example 3

```
DateTime? expiry = DateTime.Now.AddHours(1);
PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000,
            Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order comment");
```

Example 4

```
protected override void OnStart()
{
    PlaceStopOrderAsync(TradeType.Buy, Symbol, 20000, Symbol.Ask, StopOrderOnPlaced);
}
private void StopOrderOnPlaced(TradeResult tradeResult)
{
    Print("Stop order placed {0}", tradeResult.PendingOrder.Label);
}
```

Example 5

```
protected override void OnStart()
{
    bool hasTrailingStop = true;
    DateTime? expiry = DateTime.Now.AddHours(1);
    PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000,
                        Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry, "order
comment", hasTrailingStop);
}
private void StopOrderOnPlaced(TradeResult tradeResult)
{
    Print("Stop order placed with HasTrailingStop: {0}",
```

```
tradeResult.PendingOrder.HasTrailingStop);
}
```

Example 6

```
protected override void OnStart()
{
    bool hasTrailingStop = true;
    DateTime? expiry = DateTime.Now.AddHours(1);
    PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000,
                        Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry,
                        "order comment", hasTrailingStop, StopTriggerMethod.Trade);
}
private void StopOrderOnPlaced(TradeResult tradeResult)
{
    Print("Stop order placed with stop loss trigger method: {0}",
tradeResult.PendingOrder.StopLossTriggerMethod);
}
```

Example 7

```
protected override void OnStart()
{
    bool hasTrailingStop = true;
    StopTriggerMethod stopLossTriggerMethod = StopTriggerMethod.Trade;
    StopTriggerMethod stopOrderTriggerMethod = StopTriggerMethod.Trade;
    DateTime? expiry = DateTime.Now.AddHours(1);
    PlaceStopOrderAsync(TradeType.Sell, Symbol, 10000,
                        Symbol.Bid - 10* Symbol.PipSize,"myLabel", 10, 10, expiry,
                        "order comment", hasTrailingStop, stopLossTriggerMethod,
stopOrderTriggerMethod);
}
private void StopOrderOnPlaced(TradeResult tradeResult)
{
    Print("Stop order placed with stop order trigger method: {0}",
tradeResult.PendingOrder.StopOrderTriggerMethod);
}
```

PlaceStopLimitOrder

Summary

Place a Stop Limit Order

Syntax

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopLimitOrder

Summary

Place a Stop Limit Order

Syntax

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopLimitOrder

Summary

Place a Stop Limit Order

Syntax

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopLimitOrder

Summary

Place a Stop Limit Order

Syntax

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopLimitOrder

Summary

Place a Stop Limit Order

Syntax

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
```

```
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopLimitOrder

Summary

Place a Stop Limit Order

Syntax

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopLimitOrder

Summary

Place a Stop Limit Order

Syntax

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopLimitOrder

Summary

Place a Stop Limit Order

Syntax

```
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod)
```

```
public TradeResult PlaceStopLimitOrder(TradeType tradeType, Symbol symbol, double volume,
double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double?
takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod?
stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod)
```

Parameters

Name	Description

PlaceStopLimitOrderAsync

Summary

Place Stop Limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, [optional] Action
callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, [optional]
Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
```

```
StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

PlaceStopLimitOrderAsync

Summary

Place Stop Limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, [optional] Action
callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, [optional]
Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

PlaceStopLimitOrderAsync

Summary

Place Stop Limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, [optional] Action
callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
```

```
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, [optional]
Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

PlaceStopLimitOrderAsync

Summary

Place Stop Limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, [optional] Action
callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
```

```
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, [optional]
Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

PlaceStopLimitOrderAsync

Summary

Place Stop Limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
```

```
volume, double targetPrice, double stopLimitRangePips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, [optional] Action
callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, [optional]
Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

PlaceStopLimitOrderAsync

Summary

Place Stop Limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, string label, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
```

```
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

PlaceStopLimitOrderAsync

Summary

Place Stop Limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, [optional] Action
callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, [optional]
Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop,
StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod, [optional]
Action callback)
```

Parameters

Name	Description

PlaceStopLimitOrderAsync

Summary

Place Stop Limit order in asynchronous execution mode

Syntax

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, [optional] Action
callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double
volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips,
double? takeProfitPips, DateTime? expiration, [optional] Action callback)
```

```
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation PlaceStopLimitOrderAsync(TradeType tradeType, Symbol symbol, double volume, double targetPrice, double stopLimitRangePips, string label, double? stopLossPips, double? takeProfitPips, DateTime? expiration, string comment, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod stopOrderTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

CancelPendingOrderAsync

Summary

Cancel a Pending Order in asynchronous execution mode

Syntax

```
public TradeOperation CancelPendingOrderAsync(PendingOrder pendingOrder, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
if (PendingOrders.Count > 0)
{
    var pendingOrder = PendingOrders[0];
    CancelPendingOrderAsync(pendingOrder);
}
```

ModifyPendingOrderAsync

Summary

Modify a Pending Order in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, [optional] Action
callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice);
}
```

ModifyPendingOrderAsync

Summary

Modify a Pending Order in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, [optional] Action
callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice, 10, order.TakeProfitPips);
```

```
}
```

ModifyPendingOrderAsync

Summary

Modify a Pending Order in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, [optional] Action
callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime);
}
```

ModifyPendingOrderAsync

Summary

Modify a Pending Order in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, [optional] Action
callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime, 5);
}
```

ModifyPendingOrderAsync

Summary

Modify a Pending Order in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, [optional] Action
callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
```

```
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime, 5);
}
```

ModifyPendingOrderAsync

Summary

Modify a Pending Order in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, [optional] Action
callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime, 5, hasTrailingStop, StopTriggerMethod.Trade,
StopTriggerMethod.Opposite);
}
```

ModifyPendingOrderAsync

Summary

Modify a Pending Order in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, [optional] Action
callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
```

```
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime, 5, hasTrailingStop, StopTriggerMethod.Trade);

}
```

ModifyPendingOrderAsync

Summary

Modify a Pending Order in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, [optional] Action
callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime, 5, hasTrailingStop);
}
```

ModifyPendingOrderAsync

Summary

Modify a Pending Order in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, [optional] Action
callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, long volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume,
[optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
```

```
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPendingOrderAsync(PendingOrder pendingOrder, double targetPrice,
double? stopLossPips, double? takeProfitPips, DateTime? expirationTime, double volume, bool
hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, StopTriggerMethod?
stopOrderTriggerMethod, double? stopLimitRangePips, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime);
}
```

Example 2

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime, hasTrailingStop);
}
```

Example 3

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime, hasTrailingStop, StopTriggerMethod.Trade);

}
```

Example 4

```
bool hasTrailingStop = false;
foreach (var order in PendingOrders)
{
    if (order.StopLossPips == null)
        ModifyPendingOrderAsync(order, order.TargetPrice, 10, order.TakeProfitPips,
                        order.ExpirationTime, hasTrailingStop, 5, StopTriggerMethod.Trade,
StopTriggerMethod.Opposite);
}
```

ReversePositionAsync

Summary

Modify Position in asynchronous execution mode

Syntax

```
public TradeOperation ReversePositionAsync(Position position, [optional] Action callback)
```

```
public TradeOperation ReversePositionAsync(Position position, double volume, [optional] Action
callback)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null)
{
    ReversePositionAsync(position, TradeType.Sell);
}
```

ModifyPositionAsync

Summary

Modify Position in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPositionAsync(Position position, double volume, [optional] Action callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double? takeProfit, [optional] Action callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double? takeProfit, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double? takeProfit, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action callback)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null)
{
    ModifyPositionAsync(position, 20000);
}
```

ReversePositionAsync

Summary

Modify Position in asynchronous execution mode

Syntax

```
public TradeOperation ReversePositionAsync(Position position, [optional] Action callback)
```

```
public TradeOperation ReversePositionAsync(Position position, double volume, [optional] Action
callback)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null)
{
    ReversePositionAsync(position, 20000);
}
```

ModifyPositionAsync

Summary

Modify Position in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPositionAsync(Position position, double volume, [optional] Action
callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double?
takeProfit, [optional] Action callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double?
takeProfit, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double?
takeProfit, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action
callback)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null)
{
    double? stopLoss = Symbol.Ask- 10*Symbol.PipSize;
    double? takeProfit = Symbol.Ask + 10 * Symbol.PipSize;
    ModifyPositionAsync(position, stopLoss,  takeProfit);
}
```

ModifyPositionAsync

Summary

Modify Position in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPositionAsync(Position position, double volume, [optional] Action
callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double?
takeProfit, [optional] Action callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double?
takeProfit, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double?
takeProfit, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action
callback)
```

Parameters

Name	Description

Example 1

```

```

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null)
{
    double? stopLoss = Symbol.Ask- 10*Symbol.PipSize;
    double? takeProfit = Symbol.Ask + 10 * Symbol.PipSize;
    ModifyPositionAsync(position, stopLoss,  takeProfit);
}
```

Example 2

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null)
{
    double? stopLoss = Symbol.Ask- 10*Symbol.PipSize;
    double? takeProfit = Symbol.Ask + 10 * Symbol.PipSize;
    bool hasTrailingStop = true;
    ModifyPositionAsync(position, stopLoss,  takeProfit, hasTrailingStop);
}
```

ModifyPositionAsync

Summary

Modify Position in asynchronous execution mode

Syntax

```
public TradeOperation ModifyPositionAsync(Position position, double volume, [optional] Action
callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double?
takeProfit, [optional] Action callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double?
takeProfit, bool hasTrailingStop, [optional] Action callback)
```

```
public TradeOperation ModifyPositionAsync(Position position, double? stopLoss, double?
takeProfit, bool hasTrailingStop, StopTriggerMethod? stopLossTriggerMethod, [optional] Action
callback)
```

Parameters

Name	Description

Example 1

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null)
{
    double? stopLoss = Symbol.Ask- 10*Symbol.PipSize;
    double? takeProfit = Symbol.Ask + 10 * Symbol.PipSize;
    ModifyPositionAsync(position, stopLoss,  takeProfit);
}
```

Example 2

```
var position = Positions.Find("myLabel", Symbol, TradeType.Buy);
if (position != null)
{
    double? stopLoss = Symbol.Ask- 10*Symbol.PipSize;
    double? takeProfit = Symbol.Ask + 10 * Symbol.PipSize;
    bool hasTrailingStop = true;
    ModifyPositionAsync(position, stopLoss,  takeProfit, hasTrailingStop);
}
```

ToString

Summary

Returns the cBot class name

Syntax

```
public override string ToString()
```

Example 1

```
protected override void OnStart()
{
    Print(ToString());
}
```

GetFitness

Summary

Override this method to provide custom fitness value for Optimization

Syntax

```
protected virtual double GetFitness(GetFitnessArgs args)
```

Parameters

Name	Description

RobotAttribute

Summary

Sealed Class RobotAttribute.

Remarks

Marks a class as a Robot. The Robot attribute cannot be ommited.

Syntax

```
public sealed class RobotAttribute : Attribute
```

Members

Name	Type	Summary
AccessRights (2)	Property	AccessRights required for cBot
Name (5)	Property	The name of a robot. Sets from constructor.
RobotAttribute	Method	Initializes a new RobotAttribute instance and sets the name
TimeZone (3)	Property	Sets the timezone for all the robot or indicator datetime references

Name

Summary

The name of a robot. Sets from constructor.

Syntax

```
public string Name{ get; }
```

Example 1

```
[Robot("newRobot")]    //  newRobot is the name of the Robot
public class myRobot : Robot
{
    //...
}
```

TimeZone

Summary

Sets the timezone for all the robot or indicator datetime references

Remarks

All dates and times within the robot or indicator will be converted to this timezone

Syntax

```
public string TimeZone{ get; set; }
```

Example 1

```
[Robot(TimeZone = TimeZones.EasternStandardTime)]
public class NewsRobot : Robot
```

AccessRights

Summary

AccessRights required for cBot

Syntax

```
public AccessRights AccessRights{ get; set; }
```

RobotAttribute

Summary

Initializes a new RobotAttribute instance and sets the name

Remarks

Marks a class as a Robot. The Robot attribute cannot be ommited. To make it effective apply enclosed in square brackets, e.g. [Robot("Name")], before the Robot class declaration.

Syntax

```
public RobotAttribute RobotAttribute(string name)
```

```
public RobotAttribute RobotAttribute()
```

Parameters

Name	Description

Example 1

```
//...
[Robot("myRobot")]    //  myRobot is the name of the Robot
public class myRobot : Robot
{
    //...
}
//...
```

RobotAttribute

Summary

Initializes a new RobotAttribute instance.

Remarks

To make it effective apply enclosed in square brackets, e.g. [Robot], in front of the robot class declaration.

Syntax

```
public RobotAttribute RobotAttribute(string name)
```

```
public RobotAttribute RobotAttribute()
```

Example 1

```
[Robot] // RobotAttribute
public class NewRobot : Robot
{
    //...
}
```

RoundingMode

Summary

Rounding mode for normalizing trade volume

Syntax

```
public sealed enum RoundingMode
```

Members

Name	Type	Summary
Down (2)	Field	Round value down to tradable volume
ToNearest	Field	Round value to nearest tradable volume
Up (2)	Field	Round value up to tradable volume

Example 1

```
volume = Symbol.NormalizeVolume(volume, RoundingMode.Down);
```

ToNearest

Summary

Round value to nearest tradable volume

Syntax

```
RoundingMode.ToNearest
```

Example 1

```
var volume = Symbol.NormalizeVolume(calculatedVolume, RoundingMode.ToNearest);
```

Down

Summary

Round value down to tradable volume

Syntax

```
RoundingMode.Down
```

Example 1

```
var volume = Symbol.NormalizeVolume(calculatedVolume, RoundingMode.Down);
```

Up

Summary

Round value up to tradable volume

Syntax

```
RoundingMode.Up
```

Example 1

```
var volume = Symbol.NormalizeVolume(calculatedVolume, RoundingMode.Up);
```

RunningMode

Summary

Defines if a cBot is running in real time, in the silent backtesting mode, in the visual backtesting mode, or in the optimization mode.

Syntax

```
public sealed enum RunningMode
```

Members

Name	Type	Summary
Optimization	Field	The cBot is running in the optimization mode.
RealTime	Field	The cBot is running in real time.
SilentBacktesting	Field	The cBot is running in the silent backtesting mode.
VisualBacktesting	Field	The cBot is running in the visual backtesting mode.

RealTime

Summary

The cBot is running in real time.

Syntax

```
RunningMode.RealTime
```

SilentBacktesting

Summary

The cBot is running in the silent backtesting mode.

Syntax

```
RunningMode.SilentBacktesting
```

VisualBacktesting

Summary

The cBot is running in the visual backtesting mode.

Syntax

```
RunningMode.VisualBacktesting
```

Optimization

Summary

The cBot is running in the optimization mode.

Syntax

```
RunningMode.Optimization
```

StopTriggerMethod

Summary

Trigger side for Stop Orders

Syntax

```
public sealed enum StopTriggerMethod
```

Members

Name	Type	Summary
DoubleOpposite	Field	Uses opposite prices for order triggering, and waits for additional confirmation - two consecutive prices should meet criteria to trigger order. Buy order and Stop Loss for Sell position will be triggered when two consecutive Bid prices >= order price. Sell order and Stop Loss for Buy position will be triggered when two consecutive Ask prices <= order price.
DoubleTrade	Field	Uses default prices for order triggering, but waits for additional confirmation - two consecutive prices should meet criteria to trigger order. Buy order and Stop Loss for Sell position will be triggered when two consecutive Ask prices >= order price. Sell order and Stop Loss for Buy position will be triggered when two consecutive Bid prices <= order price.
Opposite	Field	Opposite method uses opposite price for order triggering. Buy order and Stop Loss for Sell position will be triggered when Bid >= order price. Sell order and Stop Loss for Buy position will be triggered when Ask <= order price.
Trade	Field	Trade method uses default trigger behavior for Stop orders. Buy order and Stop Loss for Sell position will be triggered when Ask >= order price. Sell order and Stop Loss for Buy position will be triggered when Bid <= order price.

Trade

Summary

Trade method uses default trigger behavior for Stop orders. Buy order and Stop Loss for Sell position will be triggered when Ask >= order price. Sell order and Stop Loss for Buy position will be triggered when Bid <= order price.

Syntax

```
StopTriggerMethod.Trade
```

Opposite

Summary

Opposite method uses opposite price for order triggering. Buy order and Stop Loss for Sell position will be triggered when Bid >= order price. Sell order and Stop Loss for Buy position will be triggered when Ask <= order price.

Syntax

```
StopTriggerMethod.Opposite
```

DoubleTrade

Summary

Uses default prices for order triggering, but waits for additional confirmation - two consecutive prices should meet criteria to trigger order. Buy order and Stop Loss for Sell position will be triggered when two consecutive Ask prices >= order price. Sell order and Stop Loss for Buy position will be triggered when two consecutive Bid prices <= order price.

Syntax

```
StopTriggerMethod.DoubleTrade
```

DoubleOpposite

Summary

Uses opposite prices for order triggering, and waits for additional confirmation - two consecutive prices should meet criteria to trigger order. Buy order and Stop Loss for Sell position will be triggered when two consecutive Bid prices >= order price. Sell order and Stop Loss for Buy position will be triggered when two consecutive Ask prices <= order price.

Syntax

```
StopTriggerMethod.DoubleOpposite
```

TimeFrame

Summary

Contains supported timeframe values from Minute 1 to Monthly.

Syntax

```
public class TimeFrame : Object
```

Members

Name	Type	Summary
Daily	Field	Daily Timeframe
Day2	Field	2 day Timeframe
Day3	Field	3 day Timeframe
Hour	Field	1 hour Timeframe
Hour12	Field	12 hour Timeframe
Hour2	Field	2 hour Timeframe
Hour3	Field	3 hour Timeframe
Hour4	Field	4 hour Timeframe
Hour6	Field	6 hour Timeframe
Hour8	Field	8 hour Timeframe
Minute	Field	1 Minute Timeframe
Minute10	Field	10 Minute Timeframe
Minute15	Field	15 Minute Timeframe
Minute2	Field	2 Minute Timeframe
Minute20	Field	20 Minute Timeframe
Minute3	Field	3 Minute Timeframe
Minute30	Field	30 Minute Timeframe
Minute4	Field	4 Minute Timeframe
Minute45	Field	45 Minute Timeframe
Minute5	Field	5 Minute Timeframe
Minute6	Field	6 Minute Timeframe
Minute7	Field	7 Minute Timeframe
Minute8	Field	8 Minute Timeframe
Minute9	Field	9 Minute Timeframe
Monthly	Field	Monthly Timeframe

ToString (4)	Method	Convert the TimeFrame property to a string
Weekly	Field	Weekly Timeframe

Example 1

```
if(TimeFrame  < TimeFrame.Daily)
    Print("Intraday Trading");
```

ToString

Summary

Convert the TimeFrame property to a string

Syntax

```
public override string ToString()
```

Example 1

```
Print("TimeFrame is {0}", TimeFrame.Daily.ToString());
```

Minute

Summary

1 Minute Timeframe

Syntax

```
public static TimeFrame Minute
```

Minute2

Summary

2 Minute Timeframe

Syntax

```
public static TimeFrame Minute2
```

Minute3

Summary

3 Minute Timeframe

Syntax

```
public static TimeFrame Minute3
```

Minute4

Summary

4 Minute Timeframe

Syntax

```
public static TimeFrame Minute4
```

Minute5

Summary

5 Minute Timeframe

Syntax

```
public static TimeFrame Minute5
```

Minute6

Summary

6 Minute Timeframe

Syntax

```
public static TimeFrame Minute6
```

Minute7

Summary

7 Minute Timeframe

Syntax

```
public static TimeFrame Minute7
```

Minute8

Summary

8 Minute Timeframe

Syntax

```
public static TimeFrame Minute8
```

Minute9

Summary

9 Minute Timeframe

Syntax

```
public static TimeFrame Minute9
```

Minute10

Summary

10 Minute Timeframe

Syntax

```
public static TimeFrame Minute10
```

Minute15

Summary

15 Minute Timeframe

Syntax

```
public static TimeFrame Minute15
```

Minute20

Summary

20 Minute Timeframe

Syntax

```
public static TimeFrame Minute20
```

Minute30

Summary

30 Minute Timeframe

Syntax

```
public static TimeFrame Minute30
```

Minute45

Summary

45 Minute Timeframe

Syntax

```
public static TimeFrame Minute45
```

Hour

Summary

1 hour Timeframe

Syntax

```
public static TimeFrame Hour
```

Hour2

Summary

2 hour Timeframe

Syntax

```
public static TimeFrame Hour2
```

Hour3

Summary

3 hour Timeframe

Syntax

```
public static TimeFrame Hour3
```

Hour4

Summary

4 hour Timeframe

Syntax

```
public static TimeFrame Hour4
```

Hour6

Summary

6 hour Timeframe

Syntax

```
public static TimeFrame Hour6
```

Hour8

Summary

8 hour Timeframe

Syntax

```
public static TimeFrame Hour8
```

Hour12

Summary

12 hour Timeframe

Syntax

```
public static TimeFrame Hour12
```

Daily

Summary

Daily Timeframe

Syntax

```
public static TimeFrame Daily
```

Day2

Summary

2 day Timeframe

Syntax

```
public static TimeFrame Day2
```

Day3

Summary

3 day Timeframe

Syntax

```
public static TimeFrame Day3
```

Weekly

Summary

Weekly Timeframe

Syntax

```
public static TimeFrame Weekly
```

Monthly

Summary

Monthly Timeframe

Syntax

```
public static TimeFrame Monthly
```

Timer

Summary

Schedules execution of virtual OnTimer method with specified interval.

Syntax

```
public interface Timer
```

Members

Name	Type	Summary
Interval	Property	Gets the interval of timer. Returns -1 millisecond if the timer is stopped
Start	Method	Starts the Timer
Stop (3)	Method	Stops the Timer
TimerTick	Event	Occurs when the interval elapses

Interval

Summary

Gets the interval of timer. Returns -1 millisecond if the timer is stopped

Syntax

```
public TimeSpan Interval{ get; }
```

Start

Summary

Starts the Timer

Syntax

```
public void Start(TimeSpan interval)
```

```
public void Start(int intervalInSeconds)
```

Parameters

Name	Description

Start

Summary

Starts the Timer

Syntax

```
public void Start(TimeSpan interval)
```

```
public void Start(int intervalInSeconds)
```

Parameters

Name	Description

Stop

Summary

Stops the Timer

Syntax

```
public void Stop()
```

TimerTick

Summary

Occurs when the interval elapses

Syntax

```
public event Action TimerTick
```

Example 1

```
protected override void OnStart()
{
    Timer.TimerTick += OnTimerTick
    Timer.Start(1);//start timer with 1 second interval
}
private void OnTimerTick()
{
    ChartObjects.DrawText("time", Time.ToString("HH:mm:ss"), StaticPosition.TopLeft);
}
```

TimeSeries

Summary

A series of values that represent time like MarketSeries.OpenTime

Syntax

```
public interface TimeSeries
```

Members

Name	Type	Summary
Count (6)	Property	Gets the number of elements contained in the series.

GetIndexByExactTime	Method	Find the index in the different time frame series.
GetIndexByTime	Method	Find the index in the different time frame series.
Last (2)	Method	Access a value in the data series certain number of bars ago.
LastValue (2)	Property	Gets the last value of this time series.
this[int index] (7)	Property	Returns the DateTime value at the specified index.

this[int index]

Summary

Returns the DateTime value at the specified index.

Syntax

```
public DateTime this[int index]{ get; }
```

Parameters

Name	Description

LastValue

Summary

Gets the last value of this time series.

Syntax

```
public DateTime LastValue{ get; }
```

Example 1

```
DateTime openTime = MarketSeries.OpenTime.LastValue;
```

Count

Summary

Gets the number of elements contained in the series.

Syntax

```
public int Count{ get; }
```

Last

Summary

Access a value in the data series certain number of bars ago.

Syntax

```
public DateTime Last(int index)
```

Parameters

Name	Description

Example 1

```
DateTime openTime = MarketSeries.OpenTime.Last[5];
```

GetIndexByExactTime

Summary

Find the index in the different time frame series.

Syntax

```
public int GetIndexByExactTime(DateTime dateTime)
```

Parameters

Name	Description

Example 1

```
var indexSeries2 = indexSeries2.OpenTime.GetIndexByExactTime(MarketSeries.OpenTime.LastValue);
```

GetIndexByTime

Summary

Find the index in the different time frame series.

Syntax

```
public int GetIndexByTime(DateTime dateTime)
```

Parameters

Name	Description

Example 1

```
var indexSeries2 = indexSeries2.OpenTime.GetIndexByTime(MarketSeries.OpenTime.LastValue);
```

TimeZones

Summary

Standard TimeZones Class

Remarks

Sets the timezone for all the robot or indicator datetime references

Syntax

```
public static sealed class TimeZones : Object
```

Members

Name	Type	Summary
AlaskanStandardTime	Field	(GMT-09:00) Alaska
ArabianStandardTime	Field	(GMT+04:00) Abu Dhabi, Muscat
ArabicStandardTime	Field	(GMT+03:00) Baghdad
ArabStandardTime	Field	(GMT+03:00) Kuwait, Riyadh
AtlanticStandardTime	Field	(GMT-04:00) Atlantic Time (Canada)
AzoresStandardTime	Field	(GMT-01:00) Azores
CanadaCentralStandardTime	Field	(GMT-06:00) Saskatchewan
CapeVerdeStandardTime	Field	(GMT-01:00) Cape Verde Islands
CaucasusStandardTime	Field	(GMT+04:00) Baku, Tbilisi, Yerevan
CenAustraliaStandardTime	Field	(GMT+09:30) Adelaide
CentralAmericaStandardTime	Field	(GMT-06:00) Central America
CentralAsiaStandardTime	Field	(GMT+06:00) Astana, Dhaka
CentralEuropeanStandardTime	Field	(GMT+01:00) Sarajevo, Skopje, Warsaw, Zagreb
CentralEuropeStandardTime	Field	(GMT+01:00) Belgrade, Bratislava, Budapest, Ljubljana, Prague
CentralPacificStandardTime	Field	(GMT+11:00) Magadan, Solomon Islands, New Caledonia
CentralStandardTime	Field	(GMT-06:00) Central Time (US and Canada
ChinaStandardTime	Field	(GMT+08:00) Beijing, Chongqing, Hong Kong SAR, Urumqi
DatelineStandardTime	Field	(GMT-12:00) International Date Line West
EAfricaStandardTime	Field	(GMT+03:00) Nairobi
EasternStandardTime	Field	(GMT-05:00) Eastern Time (US and Canada)
EAustraliaStandardTime	Field	(GMT+10:00) Brisbane
EEuropeStandardTime	Field	(GMT+02:00) Bucharest
EgyptStandardTime	Field	(GMT+02:00) Cairo
EkaterinburgStandardTime	Field	(GMT+05:00) Ekaterinburg
ESouthAmericaStandardTime	Field	(GMT-03:00) Brasilia
FLEStandardTime	Field	(GMT+02:00) Helsinki, Kiev, Riga, Sofia, Tallinn, Vilnius
GMTStandardTime	Field	(GMT) Greenwich Mean Time: Dublin, Edinburgh, Lisbon, London
GreenlandStandardTime	Field	(GMT-03:00) Greenland
GreenwichStandardTime	Field	(GMT) Casablanca, Monrovia
GTBStandardTime	Field	(GMT+02:00) Athens, Istanbul, Minsk
HawaiianStandardTime	Field	(GMT-10:00) Hawaii

IndiaStandardTime	Field	(GMT+05:30) Chennai, Kolkata, Mumbai, New Delhi
IranStandardTime	Field	(GMT+03:30) Tehran
IsraelStandardTime	Field	(GMT+02:00) Jerusalem
KoreaStandardTime	Field	(GMT+09:00) Seoul
MidAtlanticStandardTime	Field	(GMT-02:00) Mid-Atlantic
MountainStandardTime	Field	(GMT-07:00) Mountain Time (US and Canada)
MyanmarStandardTime	Field	(GMT+06:30) Yangon Rangoon
NCentralAsiaStandardTime	Field	(GMT+06:00) Almaty, Novosibirsk
NepalStandardTime	Field	(GMT+05:45) Kathmandu
NewZealandStandardTime	Field	(GMT+12:00) Auckland, Wellington
NorthAsiaEastStandardTime	Field	(GMT+08:00) Irkutsk, Ulaanbaatar
NorthAsiaStandardTime	Field	(GMT+07:00) Krasnoyarsk
PacificStandardTime	Field	(GMT-08:00) Pacific Time (US and Canada); Tijuana
RomanceStandardTime	Field	(GMT+01:00) Brussels, Copenhagen, Madrid, Paris
RussianStandardTime	Field	(GMT+03:00) Moscow, St. Petersburg, Volgograd
SamoaStandardTime	Field	(GMT-11:00) Midway Island, Samoa
SingaporeStandardTime	Field	(GMT+08:00) Kuala Lumpur, Singapore
SouthAfricaStandardTime	Field	(GMT+02:00) Harare, Pretoria
SriLankaStandardTime	Field	(GMT+06:00) Sri Jayawardenepura
TaipeiStandardTime	Field	(GMT+08:00) Taipei
TasmaniaStandardTime	Field	(GMT+10:00) Hobart
TokyoStandardTime	Field	(GMT+09:00) Osaka, Sapporo, Tokyo
TongaStandardTime	Field	(GMT+13:00) Nuku'alofa
UTC	Field	Coordinated Universal Time
VladivostokStandardTime	Field	(GMT+10:00) Vladivostok
WAustraliaStandardTime	Field	(GMT+08:00) Perth
WCentralAfricaStandardTime	Field	(GMT+01:00) West Central Africa
WestAsiaStandardTime	Field	(GMT+05:00) Islamabad, Karachi, Tashkent
WestPacificStandardTime	Field	(GMT+10:00) Guam, Port Moresby
WEuropeStandardTime	Field	(GMT+01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
YakutskStandardTime	Field	(GMT+09:00) Yakutsk

Example 1

```
```

```
[Robot(TimeZone = TimeZones.EasternStandardTime)]
public class NewsRobot : Robot
```

DatelineStandardTime

Summary

(GMT-12:00) International Date Line West

Syntax

```
public static string DatelineStandardTime
```

SamoaStandardTime

Summary

(GMT-11:00) Midway Island, Samoa

Syntax

```
public static string SamoaStandardTime
```

HawaiianStandardTime

Summary

(GMT-10:00) Hawaii

Syntax

```
public static string HawaiianStandardTime
```

AlaskanStandardTime

Summary

(GMT-09:00) Alaska

Syntax

```
public static string AlaskanStandardTime
```

PacificStandardTime

Summary

(GMT-08:00) Pacific Time (US and Canada); Tijuana

Syntax

```
public static string PacificStandardTime
```

MountainStandardTime

Summary

(GMT-07:00) Mountain Time (US and Canada)

Syntax

```
public static string MountainStandardTime
```

CentralStandardTime

Summary

(GMT-06:00) Central Time (US and Canada

Syntax

```
public static string CentralStandardTime
```

CanadaCentralStandardTime

Summary

(GMT-06:00) Saskatchewan

Syntax

```
public static string CanadaCentralStandardTime
```

CentralAmericaStandardTime

Summary

(GMT-06:00) Central America

Syntax

```
public static string CentralAmericaStandardTime
```

EasternStandardTime

Summary

(GMT-05:00) Eastern Time (US and Canada)

Syntax

```
public static string EasternStandardTime
```

AtlanticStandardTime

Summary

(GMT-04:00) Atlantic Time (Canada)

Syntax

```
public static string AtlanticStandardTime
```

ESouthAmericaStandardTime

Summary

(GMT-03:00) Brasilia

Syntax

```
public static string ESouthAmericaStandardTime
```

GreenlandStandardTime

Summary

(GMT-03:00) Greenland

Syntax

```
public static string GreenlandStandardTime
```

MidAtlanticStandardTime

Summary

(GMT-02:00) Mid-Atlantic

Syntax

```
public static string MidAtlanticStandardTime
```

AzoresStandardTime

Summary

(GMT-01:00) Azores

Syntax

```
public static string AzoresStandardTime
```

CapeVerdeStandardTime

Summary

(GMT-01:00) Cape Verde Islands

Syntax

```
public static string CapeVerdeStandardTime
```

GMTStandardTime

Summary

(GMT) Greenwich Mean Time: Dublin, Edinburgh, Lisbon, London

Syntax

```
public static string GMTStandardTime
```

GreenwichStandardTime

Summary

(GMT) Casablanca, Monrovia

Syntax

```
public static string GreenwichStandardTime
```

CentralEuropeStandardTime

Summary

(GMT+01:00) Belgrade, Bratislava, Budapest, Ljubljana, Prague

Syntax

```
public static string CentralEuropeStandardTime
```

CentralEuropeanStandardTime

Summary

(GMT+01:00) Sarajevo, Skopje, Warsaw, Zagreb

Syntax

```
public static string CentralEuropeanStandardTime
```

RomanceStandardTime

Summary

(GMT+01:00) Brussels, Copenhagen, Madrid, Paris

Syntax

```
public static string RomanceStandardTime
```

WEuropeStandardTime

Summary

(GMT+01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna

Syntax

```
public static string WEuropeStandardTime
```

WCentralAfricaStandardTime

Summary

(GMT+01:00) West Central Africa

Syntax

```
public static string WCentralAfricaStandardTime
```

EEuropeStandardTime

Summary

(GMT+02:00) Bucharest

Syntax

```
public static string EEuropeStandardTime
```

EgyptStandardTime

Summary

(GMT+02:00) Cairo

Syntax

```
public static string EgyptStandardTime
```

FLEStandardTime

Summary

(GMT+02:00) Helsinki, Kiev, Riga, Sofia, Tallinn, Vilnius

Syntax

```
public static string FLEStandardTime
```

GTBStandardTime

Summary

(GMT+02:00) Athens, Istanbul, Minsk

Syntax

```
public static string GTBStandardTime
```

IsraelStandardTime

Summary

(GMT+02:00) Jerusalem

Syntax

```
public static string IsraelStandardTime
```

SouthAfricaStandardTime

Summary

(GMT+02:00) Harare, Pretoria

Syntax

```
public static string SouthAfricaStandardTime
```

RussianStandardTime

Summary

(GMT+03:00) Moscow, St. Petersburg, Volgograd

Syntax

```
public static string RussianStandardTime
```

ArabStandardTime

Summary

(GMT+03:00) Kuwait, Riyadh

Syntax

```

```

```
public static string ArabStandardTime
```

EAfricaStandardTime

Summary

(GMT+03:00) Nairobi

Syntax

```
public static string EAfricaStandardTime
```

ArabicStandardTime

Summary

(GMT+03:00) Baghdad

Syntax

```
public static string ArabicStandardTime
```

IranStandardTime

Summary

(GMT+03:30) Tehran

Syntax

```
public static string IranStandardTime
```

ArabianStandardTime

Summary

(GMT+04:00) Abu Dhabi, Muscat

Syntax

```
public static string ArabianStandardTime
```

CaucasusStandardTime

Summary

(GMT+04:00) Baku, Tbilisi, Yerevan

Syntax

```
public static string CaucasusStandardTime
```

EkaterinburgStandardTime

Summary

(GMT+05:00) Ekaterinburg

Syntax

```
public static string EkaterinburgStandardTime
```

WestAsiaStandardTime

Summary

(GMT+05:00) Islamabad, Karachi, Tashkent

Syntax

```
public static string WestAsiaStandardTime
```

IndiaStandardTime

Summary

(GMT+05:30) Chennai, Kolkata, Mumbai, New Delhi

Syntax

```
public static string IndiaStandardTime
```

NepalStandardTime

Summary

(GMT+05:45) Kathmandu

Syntax

```
public static string NepalStandardTime
```

CentralAsiaStandardTime

Summary

(GMT+06:00) Astana, Dhaka

Syntax

```
public static string CentralAsiaStandardTime
```

SriLankaStandardTime

Summary

(GMT+06:00) Sri Jayawardenepura

Syntax

```
public static string SriLankaStandardTime
```

NCentralAsiaStandardTime

Summary

(GMT+06:00) Almaty, Novosibirsk

Syntax

```
public static string NCentralAsiaStandardTime
```

MyanmarStandardTime

Summary

(GMT+06:30) Yangon Rangoon

Syntax

```
public static string MyanmarStandardTime
```

NorthAsiaStandardTime

Summary

(GMT+07:00) Krasnoyarsk

Syntax

```
public static string NorthAsiaStandardTime
```

ChinaStandardTime

Summary

(GMT+08:00) Beijing, Chongqing, Hong Kong SAR, Urumqi

Syntax

```
public static string ChinaStandardTime
```

SingaporeStandardTime

Summary

(GMT+08:00) Kuala Lumpur, Singapore

Syntax

```
public static string SingaporeStandardTime
```

TaipeiStandardTime

Summary

(GMT+08:00) Taipei

Syntax

```
public static string TaipeiStandardTime
```

WAustraliaStandardTime

Summary

(GMT+08:00) Perth

Syntax

```
public static string WAustraliaStandardTime
```

NorthAsiaEastStandardTime

Summary

(GMT+08:00) Irkutsk, Ulaanbaatar

Syntax

```
public static string NorthAsiaEastStandardTime
```

KoreaStandardTime

Summary

(GMT+09:00) Seoul

Syntax

```
public static string KoreaStandardTime
```

TokyoStandardTime

Summary

(GMT+09:00) Osaka, Sapporo, Tokyo

Syntax

```
public static string TokyoStandardTime
```

YakutskStandardTime

Summary

(GMT+09:00) Yakutsk

Syntax

```
public static string YakutskStandardTime
```

CenAustraliaStandardTime

Summary

(GMT+09:30) Adelaide

Syntax

```
public static string CenAustraliaStandardTime
```

EAustraliaStandardTime

Summary

(GMT+10:00) Brisbane

Syntax

```
public static string EAustraliaStandardTime
```

TasmaniaStandardTime

Summary

(GMT+10:00) Hobart

Syntax

```
public static string TasmaniaStandardTime
```

VladivostokStandardTime

Summary

(GMT+10:00) Vladivostok

Syntax

```
public static string VladivostokStandardTime
```

WestPacificStandardTime

Summary

(GMT+10:00) Guam, Port Moresby

Syntax

```
public static string WestPacificStandardTime
```

CentralPacificStandardTime

Summary

(GMT+11:00) Magadan, Solomon Islands, New Caledonia

Syntax

```
public static string CentralPacificStandardTime
```

NewZealandStandardTime

Summary

(GMT+12:00) Auckland, Wellington

Syntax

```
public static string NewZealandStandardTime
```

TongaStandardTime

Summary

(GMT+13:00) Nuku'alofa

Syntax

```
public static string TongaStandardTime
```

UTC

Summary

Coordinated Universal Time

Syntax

```
public static string UTC
```

TradeOperation

Summary

Provides access to properties describing an asynchronous trade operation.

Syntax

```
public class TradeOperation : Object
```

Members

Name	Type	Summary
IsExecuting	Property	True if a trade operation is being executed, false if it completed
SetResult	Method	
ToString (5)	Method	The description of a trade operation
TradeOperation	Method	
TradeResult (2)	Property	The result of a trade operation

Example 1

```
TradeOperation operation = ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000,
"asynchronous");
if (operation.IsExecuting)
{
    Print("Trade is executing");
}
else
{
    if (operation.TradeResult.IsSuccessful)
        Print("Trade executed");
}
```

Example 2

```
protected override void OnStart()
{
    Positions.Opened += PositionsOnOpened;
    TradeOperation operation = ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000,
"asynchronous");
```

```
    ExecuteMarketOrder(TradeType.Buy, Symbol, 10000, "synchronous", 10, 10);
    if (operation.IsExecuting)
    {
        Print("Trade is executing");
    }
    else
    {
        if (operation.TradeResult.IsSuccessful)
            Print("Trade executed");
    }
}
private void PositionsOnOpened(PositionOpenedEventArgs args)
{
    var pos = args.Position;
    Print("Position {0} opened at {1}", pos.Label, pos.EntryPrice);
}
```

IsExecuting

Summary

True if a trade operation is being executed, false if it completed

Syntax

```
public bool IsExecuting{ get; }
```

Example 1

```
TradeOperation operation = ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 20000, "myLabel");
// ...
if (!operation.IsExecuting)
{
    Print("Trade executed");
}
```

TradeResult

Summary

The result of a trade operation

Syntax

```
public TradeResult TradeResult{ get; }
```

Example 1

```
TradeOperation operation = ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 20000, "myLabel");
// ...
if (!operation.IsExecuting && operation.TradeResult.IsSuccessful)
{
    Print("Trade {0} executed", operation.TradeResult.Position.Label);
}
```

TradeOperation

Syntax

```
protected TradeOperation TradeOperation(string shortDescription, TradeResult tradeResult)
```

Parameters

Name	Description

SetResult

Syntax

```
protected void SetResult(TradeResult tradeResult)
```

Parameters

Name	Description

ToString

Summary

The description of a trade operation

Syntax

```
public override string ToString()
```

Example 1

```
TradeOperation operation = ExecuteMarketOrderAsync(TradeType.Buy, Symbol, 10000,
"asynchronous");
Print(operation.ToString());
```

TradeResult

Summary

The result of a trade operation

Syntax

```
public class TradeResult : Object
```

Members

Name	Type	Summary
Error	Property	Error code of un unsuccessful trade
IsSuccessful	Property	True if the trade is successful, false if there is an error
PendingOrder (5)	Property	The resulting pending order of a trade request
Position (5)	Property	The resulting position of a trade request
ToString (6)	Method	The description of a trade result
TradeResult (3)	Method	

Example 1

```
TradeResult result = ExecuteMarketOrder(TradeType.Sell, Symbol, 20000);
if (result.IsSuccessful)
    Print("Sell at {0}", result.Position.EntryPrice);
```

IsSuccessful

Summary

True if the trade is successful, false if there is an error

Syntax

```
public bool IsSuccessful{ get; }
```

Example 1

```
TradeResult result = ExecuteMarketOrder(TradeType.Buy, Symbol, 20000);
if (result.IsSuccessful)
    Print("Buy at {0}", result.Position.EntryPrice);
```

Error

Summary

Error code of un unsuccessful trade

Syntax

```
public ErrorCode? Error{ get; }
```

Example 1

```
var mySymbol = MarketData.GetSymbol("EURUSD");
TradeResult result = ExecuteMarketOrder(TradeType.Sell, mySymbol, 1);
if(!result.IsSuccessful)
    Print("Error: {0}", result.Error);
```

Position

Summary

The resulting position of a trade request

Syntax

```
public Position Position{ get; }
```

Example 1

```
TradeResult result = ExecuteMarketOrder(TradeType.Sell, Symbol, 50000);
if (result.IsSuccessful)
    Print("Sell at {0}", result.Position.EntryPrice);
```

PendingOrder

Summary

The resulting pending order of a trade request

Syntax

```
public PendingOrder PendingOrder{ get; }
```

Example 1

```
TradeResult result = PlaceLimitOrder(TradeType.Sell, Symbol,
                    50000, Symbol.Ask, "myLabel", 10, null);
if(result.IsSuccessful)
    Print("Order placed. SL: {0}", result.PendingOrder.StopLoss);
```

TradeResult

Syntax

```
protected TradeResult TradeResult(bool isSuccessfull, ErrorCode? error, Position position,
```

```
PendingOrder pendingOrder)
```

Parameters

Name	Description

ToString

Summary

The description of a trade result

Syntax

```
public override string ToString()
```

Example 1

```
TradeResult result = PlaceLimitOrder(TradeType.Sell, Symbol, 50000, Symbol.Ask);
if (result.IsSuccessful)
    Print(result.ToString());
```

TradeType

Summary

The direction of a trade order.

Remarks

Indicates the trade direction, whether it is a Buy or a Sell trade.

Syntax

```
public sealed enum TradeType
```

Members

Name	Type	Summary

Buy	Field	Represents a Buy order.
Sell	Field	Represents a Sell order.

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 20000);
```

Example 2

```
Position position =  Positions.Find("myLabel", Symbol, TradeType.Sell);
```

Example 3

```
PlaceLimitOrder(TradeType.Buy, Symbol, 10000, Symbol.Bid);
```

Buy

Summary

Represents a Buy order.

Syntax

```
TradeType.Buy
```

Example 1

```
ExecuteMarketOrder(TradeType.Buy, Symbol, 10000);
```

Example 2

```
var result = PlaceLimitOrder(TradeType.Buy, Symbol, 10000, Symbol.Bid);
```

Sell

Summary

Represents a Sell order.

Syntax

```
TradeType.Sell
```

Example 1

```
ExecuteMarketOrder(TradeType.Sell, Symbol, 10000);
```

Example 2

```
var result = PlaceLimitOrder(TradeType.Sell, Symbol, 10000, Symbol.Ask);
```

VerticalAlignment

Summary

Describes vertical position related to an anchor point or a parent element

Syntax

```
public sealed enum VerticalAlignment
```

Members

Name	Type	Summary
Bottom (4)	Field	Bottom vertical alignment.
Center (2)	Field	Center vertical alignment.
Stretch (2)	Field	
Top (4)	Field	Top vertical alignment.

Center

Summary

Center vertical alignment.

Syntax

```
VerticalAlignment.Center
```

Top

Summary

Top vertical alignment.

Syntax

```
VerticalAlignment.Top
```

Bottom

Summary

Bottom vertical alignment.

Syntax

```
VerticalAlignment.Bottom
```

Stretch

Syntax

```
VerticalAlignment.Stretch
```