ProRealTime SOFTWARE

Programmierhandbuch

Indikatoren & Grundfunktionen (ProBuilder)

ProBuilder ist eine auf BASIC basierende Programmiersprache, die es Ihnen ermöglicht, persönliche technische Indikatoren, Handelssysteme und Marktscanner für ProRealTime zu entwickeln.

Um über alle Neuheiten informiert zu sein, können Sie uns auf Google+ folgen.

V 3.0.0 – 20161222

INHALTSVERZEICHNIS

Einführung in ProBuilder ____ 1

Kapitel I: Grundlagen ____ 2

- Verwendung von ProBuilder ... 2
  - Kurze Einführung zum Erstellen von Indikatoren ... 2
  - Tastatur-Shortcuts im Programmierfenster ... 5
- Besonderheiten der ProBuilder-Programmiersprache ... 6
- Finanztechnische Konstanten in ProBuilder ... 7
  - Kurs und Volumen, je nach Zeiteinheit des Charts ... 7
  - Konstanten für Tageskurse ... 8
  - Zeitliche Konstanten ... 8
  - Vom Preis abgeleitete Konstanten ... 12
  - Die unbestimmte Konstante ... 12
- Verwenden von bereits vorhandenen Indikatoren ... 12
- Variablen ... 13

Kapitel II: Math. Funktionen und ProBuilder-Anweisungen ____ 16

- Kontrollstrukturen ... 16
  - Bedingte IF-Anweisung ... 16
    - Eine Bedingung, ein Ergebnis (IF THEN ENDIF) ... 16
    - Eine Bedingung, zwei Ergebnisse (IF THEN ELSE ENDIF) ... 16
    - Sequentielle IF-Bedingungen ... 16
    - Mehrfach-Bedingungen (IF THEN ELSE ELSIF ENDIF) ... 17
  - Iterative Schleife FOR ... 18
    - Ansteigend (FOR, TO, DO, NEXT) ... 18
    - Absteigende Folge (FOR, DOWNTO, DO, NEXT) ... 19
  - Bedingte Schleife WHILE ... 20
  - BREAK ... 21
    - Mit WHILE ... 21
    - Mit FOR ... 21
  - CONTINUE ... 22
    - Mit WHILE ... 22
    - Mit FOR ... 22
  - ONCE ... 23
- Mathematische Funktionen ... 24
  - Übliche unäre und binäre Operatoren ... 24
  - Übliche mathematische Operatoren ... 24
  - Chart-Vergleichsfunktionen ... 24
  - Summenfunktion ... 25
  - Statistische Funktionen ... 25
- Logische Operatoren ... 25

ProBuilder-Anweisungen....25
RETURN....26
REM oder //....26
CustomClose....26
CALL....27
AS....27
COLOURED....27
Progammieranweisungen zum Zeichnen....29

Kapitel III: Praktische Aspekte 32

Erstellen eines binären bzw. ternären Indikators: Wie und Warum?....32
Erstellen von STOP-Indikatoren: Überwachen Sie Ihre Positionen in Echtzeit....33
STOP mit statischer Gewinnmitnahme....34
Statischer STOP Loss....34
Inaktivitäts-STOP....35
Trailing STOP....36

Kapitel IV: Übungen 37

Candlestick-Muster....37
Indikatoren....38

Glossar 40

Hinweis: ProRealTime bietet keinerlei Dienstleistungen im Bereich finanzieller Investitionen. In diesem Dokument zur Verfügung stehende Informationen stellen in keiner Hinsicht eine Investitionsberatung oder Aufforderung zum Kaufen oder Verkaufen irgendwelcher Finanzinstrumente dar.

Die in diesem Handbuch dargestellten Beispiele haben ausschließlich pädagogischen Charakter. Bei Ihrem persönlichen Trading sind Sie bei der Wahl Ihrer Kriterien absolut unabhängig. Historische Performance ist keinesfalls eine zuverlässige Kennzahl für zukünftige Performance und ist nicht konstant.

Jedes Handelssystem kann Sie dem Risiko aussetzen, einen Ihre Einlage übersteigenden Verlust zu erleiden.

Einführung in ProBuilder

ProBuilder ist die Programmiersprache von ProRealTime. Diese ermöglicht Ihnen das Erstellen von persönlichen technischen Indikatoren, Handelsstrategien (ProBacktest) oder Marktscannern (ProScreener). Aufgrund einiger Besonderheiten der einzelnen Module gibt es für ProBacktest und ProScreener ein eigenes Handbuch.

ProBuilder ist eine auf BASIC basierende Programmiersprache, die sowohl sehr leicht zu handhaben ist, als auch viele Möglichkeiten zur Verfügung stellt.

Mithilfe der Programmieranweisungen werden Sie in der Lage sein, Ihre eigenen Programme in jedem in ProRealTime angebotenen Tool zu erstellen. Einige der grundsätzlichen Befehle sind:

- Eröffnungskurs von jedem Balken: Open
- Schlusskurs von jedem Balken: Close
- Höchstkurs von jedem Balken: High
- Tiefstkurs von jedem Balken: Low
- Volumen von jedem Balken: Volume

Balken oder Candlesticks sind übliche Chartdarstellungen bei der Verwendung von Echtzeitdaten. Natürlich haben Sie in ProRealTime auch die Möglichkeit, eine andere Chartdarstellung zu verwenden. Sie können Renko, Kagi, Heikin-Ashi und einige andere Ansichten benutzen.

ProBuilder wertet die Daten eines jeden Balkens aus; die Berechnung der Formel beginnt beim ältesten Balken, um dann den Wert der Indikatoren in Bezug auf den aktuellen Balken zu bestimmen.

Die Indikatoren, die in der Programmiersprache ProBuilder geschrieben werden, können entweder im Preis-Chart oder in einem separaten Indikator-Fenster angezeigt werden.

In diesem Handbuch erlernen Sie anhand eines leicht verständlichen Überblicks mit konkreten Beispielen Schritt für Schritt, wie man die verfügbaren Befehle anwendet, die zum Erstellen eines Programmcodes nötig sind.

Auf den letzten Seiten des Handbuchs finden Sie ein Glossar mit einer Komplettübersicht aller ProBuilder-Befehle, vorhandener Indikatoren und anderer Funktionen, die Sie in den folgenden Kapiteln kennen lernen werden.

Nutzer, die mehr Vertrauen in ihre eigenen Programmierfähigkeiten haben, können das erste Kapitel überspringen und direkt mit Kapitel II beginnen oder nur mit dem Glossar arbeiten, um schneller die benötigten Informationen zu finden.

Einsteigern empfehlen wir, sich die Video-Anleitung mit dem Titel "**Programmierung einfacher und dynamischer Indikatoren**" anzusehen und das komplette Handbuch zu lesen.

Wir wünschen Ihnen dabei viel Erfolg und viel Spaß beim Lesen!

Kapitel I: Grundlagen

Verwendung von ProBuilder

Kurze Einführung zum Erstellen von Indikatoren

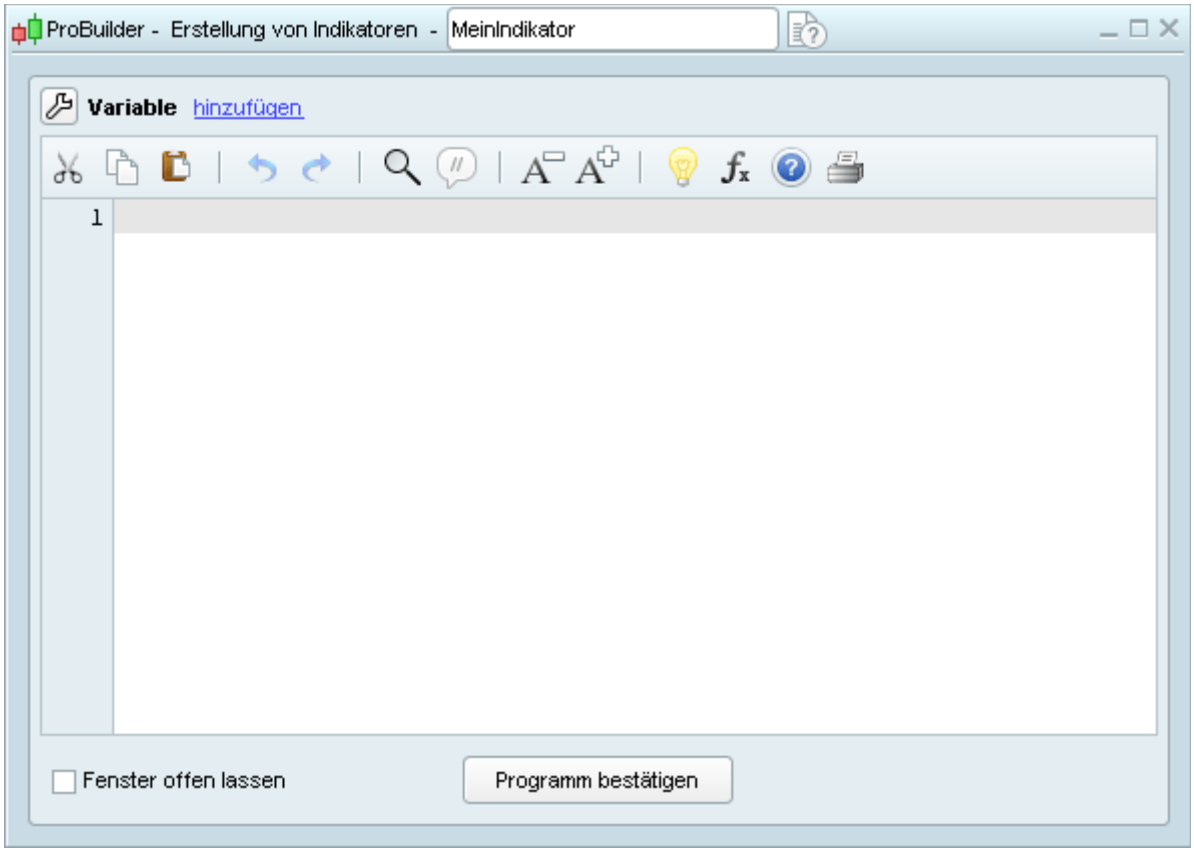
Der Programmierbereich zum Erstellen von Indikatoren öffnet sich, wenn Sie oben rechts im Chart-Fenster auf die Schaltfläche "Indikatoren/Handelssysteme" klicken.

In dem sich öffnenden Fenster der Indikatoren-Verwaltung können Sie

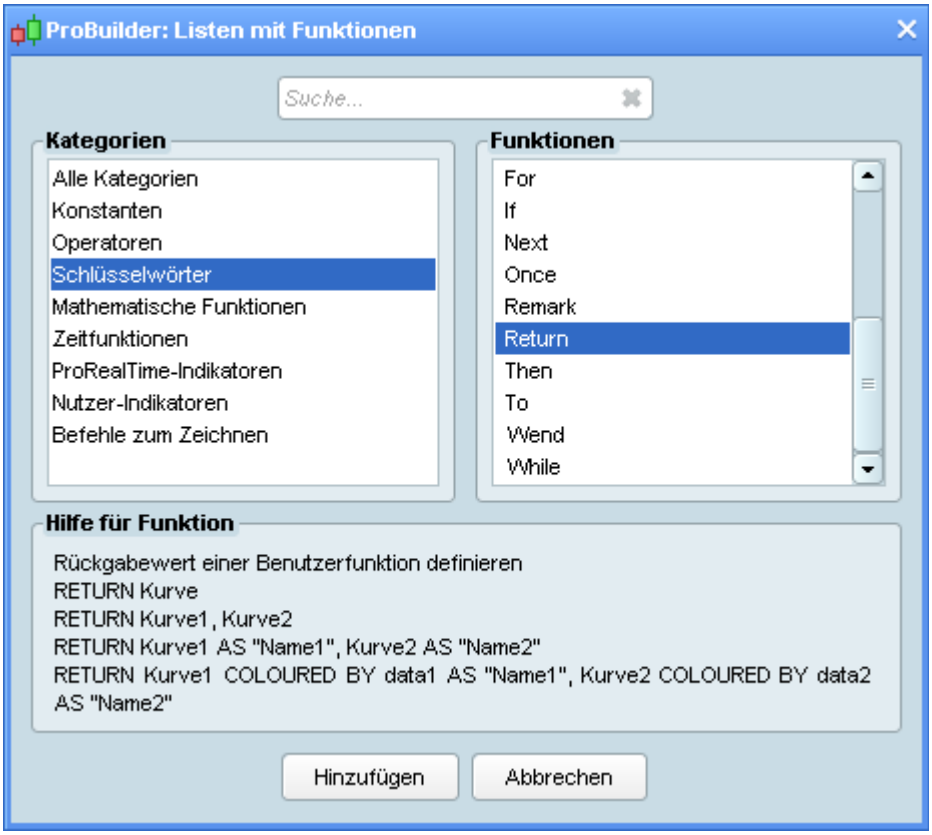
- standardmäßig vorhandene Indikatoren anzeigen,
- eigene Indikatoren erstellen, die danach auf jedes Instrument angewendet werden können.

Sollten Sie die zweite Möglichkeit wählen, dann klicken Sie auf die Schaltfläche "Neu", um Zugriff auf das Programmierfenster zu erhalten. Dort haben Sie die Wahl zwischen:

- Programmierung eines Indikators durch Eingabe des Codes direkt im Code-Editor oder
- Verwenden der Hilfe-Funktion durch Klicken auf: "Funktionen einfügen". Es öffnet sich dann ein neues Fenster, in dem alle verfügbaren Funktionen aufgelistet sind. Diese Bibliothek gliedert sich in 7 Kategorien und bietet Ihnen wichtige Hilfestellung während der Programmierung.



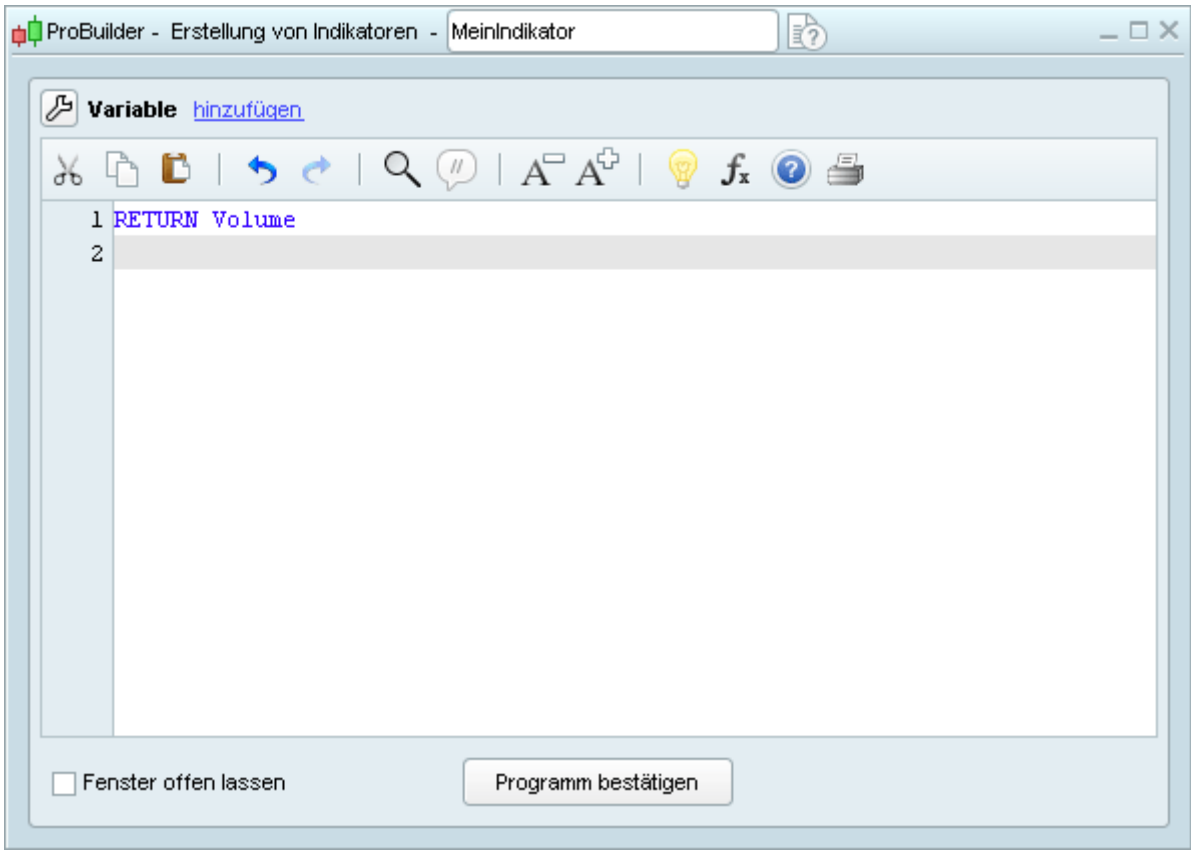
Nehmen wir z.B. das erste spezifische ProBuilder Element: die "**RETURN**" Funktion (Sie finden diese in der Kategorie "Schlüsselwörter", siehe folgende Abbildung).



Wählen Sie "**RETURN**" aus und klicken anschließend auf "Hinzufügen". Der Befehl wird dann dem Programmierfeld direkt hinzugefügt.

Mithilfe des Befehls **RETURN** wird das Ergebnis angezeigt.

Nehmen wir an, wir möchten einen das Volumen anzeigenden Indikator erstellen. Wenn Sie bereits die Funktion "**RETURN**" eingefügt haben, klicken Sie nochmals auf "Funktionen einfügen". Dann klicken Sie auf die Kategorie "Konstanten". In der rechten Spalte finden Sie die Funktion "**Volume**". Anschließend klicken Sie auf "Hinzufügen". Vergessen Sie nicht, zwischen jeder Anweisung, wie unten angezeigt, ein Leerzeichen zu setzen.



Bevor Sie auf "Programm bestätigen" klicken, müssen Sie den Namen des Indikators eingeben. In diesem Beispiel nennen wir den Indikator "VolumeDAX". Anschließend klicken Sie auf "Programm bestätigen" und der Indikator wird unter dem Preis-Chart angezeigt.

Tastatur-Shortcuts im Programmierfenster

Das Programmierfenster enthält einige praktische Funktionen, auf die mithilfe von Tastatur-Shortcuts zugegriffen werden kann (ab ProRealTime Version 10):

- Alles Auswählen (Strg + A): Den gesamten Text im Programmierfenster auswählen
- Kopieren (Strg + C): Den ausgewählten Text kopieren
- Einfügen (Strg + X): Text einfügen
- Rückgängig (Strg + Z): Die letzte im Programmierfenster getätigte Aktion rückgängig machen
- Wiederherstellen (Strg + Y): Die letzte im Programmierfenster getätigte Aktion wiederherstellen
- Suchen / Ersetzen (Strg + F): Einen Text im Programmierfenster finden / ersetzen (Groß- und Kleinschreibung wird berücksichtigt)
- Kommentare aktivieren / deaktivieren (Strg + R): Die ausgewählten Codezeilen kommentieren / Kommentar deaktivieren (kommentierter Code beginnt mit "//" oder "REM" und wird farblich grau hervorgehoben. Dieser Text wird nicht bei der Ausführung des Codes berücksichtigt).

Für Mac-Nutzer stehen die gleichen Tastatur-Shortcuts mithilfe der "Apple"-Taste anstelle der Strg-Taste zur Verfügung.

Fast alle Funktionen können auch über einen rechten Mausklick im Programmierfenster aufgerufen werden.

Besonderheiten der ProBuilder-Programmiersprache

Besonderheiten

Die ProBuilder-Sprache erlaubt Ihnen das Verwenden vieler klassischer Befehle sowie besonderer Funktionen, die speziell auf technische Analyse zugeschnitten sind. Mit diesen Befehlen lassen sich sowohl einfache als auch sehr komplexe Indikatoren programmieren.

Die Grundlagen der ProBuilder-Sprache sind folgende:

- Es ist **nicht notwendig, Variablen festzulegen.**
- Es ist **nicht notwendig, Variablen zu typisieren.**
- Es wird **kein Unterschied zwischen Groß- und Kleinschreibung gemacht.**
- **Wir verwenden das Symbol "=" sowohl für mathematisch "gleich" als auch für die Wertzuordnung von Variablen.**

Was bedeutet das ?

- Durch das Festlegen einer Variable X wird diese automatisch existent. In ProBuilder können Sie X verwenden, ohne vorher zu erklären, dass diese Variable existiert. Nehmen wir ein Beispiel:

  Mit Festlegung: Wir erstellen die Variable X, dann ordnen wir X den Wert von 5 zu.

  Ohne Festlegung: Wir ordnen der Variable X den Wert 5 zu (dadurch wird davon ausgegangen, dass X bereits existiert und ihr der Wert von 5 zugeordnet wird).

  In ProBuilder müssen Sie lediglich schreiben: X=5

- Die Typisierung einer Variable bedeutet die Festlegung ihrer Art; z.B.: Soll es sich bei der Variable um eine natürliche Zahl (z.B.: 3; 8; 21; 643; …), um eine ganze Zahl, welche positiv oder negativ sein kann (z.B.: 3; 632; -37; …), um eine Dezimalzahl (z.B.: 1,76453534535…), um eine boolesche Zahl (RICHTIG=1, FALSCH=0),…handeln?

- In ProBuilder können Sie die Befehle in großen oder in kleinen Buchstaben schreiben. Zum Beispiel kann die Gruppe der Befehle IF / THEN / ELSE / ENDIF auch folgendermaßen geschrieben werden: iF / tHeN / ELse / endIf (und auf andere Art und Weise!)

  Ausnahme: Wenn Sie sich entschließen, eine Variable zu erstellen und diese später im Programm zu verwenden, müssen Sie diese auch genau so schreiben. Wenn Sie den Variablennamen "vARiaBLe" verwendet haben und diesen in Ihrem Programm erneut verwenden wollen, dann müssen Sie auch "vARiaBLe" schreiben, nicht "variable" oder etwas anderes.

- Einer Variable kann ein Wert zugeordnet werden. Zum Verstehen dieses Prinzips müssen Sie sich die Variable wie eine leere Box vorstellen, der Sie einen Inhalt (z.B.: eine Zahl) zuordnen können. Das folgende Diagramm zeigt die Zuordnungsregel; es wird dem Wert Volumen die Variable X zugeordnet:

X ⟵ Volume

Wie Sie sehen können, müssen Sie von rechts nach links lesen: Volumen wird auf X angewendet.
Um dies in ProBuilder zu schreiben, müssen Sie nur den Pfeil durch ein Gleichheitszeichen ersetzen:

X = Volume

Das gleiche = Symbol wird verwendet:

- Zur Zuordnung einer Variable (wie im vorherigen Beispiel),
- Als mathematischer Vergleichsoperator (1+ 1= 2 ist gleich 2 = 1 + 1).

Finanztechnische Konstanten in ProBuilder

Vor der Programmierung Ihrer persönlichen Indikatoren müssen Sie zunächst die einzelnen Elemente überprüfen, die Sie zum Erstellen Ihres Codes benötigen, wie z.B. Eröffnungskurs, Schlusskurs, usw.

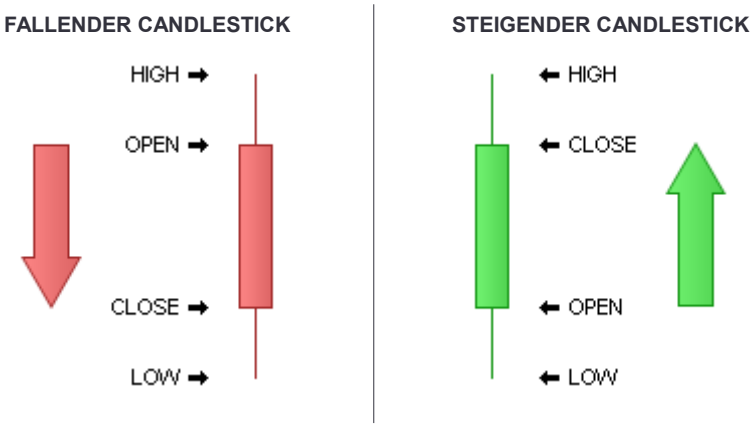
Dies sind die "Grundlagen" der technischen Analyse und die wichtigsten Dinge, die Sie über die Programmierung von Indikatoren wissen sollten.

Sie werden anschließend in der Lage sein, diese miteinander zu kombinieren, um aus den Marktdaten Informationen herauszufiltern. Wir können diese in 5 Kategorien untergliedern:

Kurs und Volumen, je nach Zeiteinheit des Charts

Das sind die "klassischen" und am meisten verwendeten Konstanten. Sie zeigen standardmäßig den Wert des aktuellen Balkens an (unabhängig von der verwendeten Zeiteinheit):

- **Open:** Eröffnungskurs von jedem Balken
- **High:** Höchstkurs von jedem Balken
- **Low:** Tiefstkurs von jedem Balken
- **Close:** Schlusskurs von jedem Balken
- **Volume:** Die Anzahl der gehandelten Aktien oder Kontrakte für jeden Balken



Beispiel: Range (Spanne) des aktuellen Balkens

```
a = High
b = Low
MyRange = a - b
RETURN MyRange
```

Wenn Sie lieber die Informationen des vorherigen Balkens anstelle des aktuellen Balkens verwenden möchten, dann müssen Sie nur die Anzahl der vorherigen Balken in eckigen Klammern eingeben (die Anzahl der Balken ab des aktuellen Balkens).

Nehmen wir als Beispiel die Konstante des Schlusskurses:

Wert des Schlusskurses des aktuellen Balkens:	*Close*
Wert des Schlusskurses vor dem aktuellen Balken:	*Close[1]*
Wert des Schlusskurses vor dem n-ten Balken vor dem aktuellen Balken:	*Close [n]*

Diese Regel ist für jede Konstante gültig. Zum Beispiel kann der Eröffnungskurs des zweiten Balkens vor dem aktuellen wie folgt ausgedrückt werden: Open[2].

Der ausgegebene Wert ist von der im Chart angezeigten Zeiteinheit abhängig.

Konstanten für Tageskurse

Im Gegensatz zu den Anweisungen, die sich auf die Zeiteinheit des Charts beziehen, berücksichtigen die Konstanten für Tageskurse den Wert eines Tages, unabhängig von der im Chart angezeigten Zeiteinheit.

Ein weiterer Unterschied zwischen den Anweisungen für tägliche Kursbedingungen und für die im Chart angezeigten Zeiteinheiten ist, dass bei Tageskursen zur Angabe des Wertes vorheriger Balken runde Klammern und nicht eckige Klammern verwendet werden.

- **DOpen(n):** Eröffnungskurs des n-ten Tages vor dem aktuellen Balken
- **DHigh(n):** Höchstkurs des n-ten Tages vor dem aktuellen Balken
- **DLow(n):** Tiefstkurs des n-ten Tages vor dem aktuellen Balken
- **DClose(n):** Schlusskurs des n-ten Tages vor dem aktuellen Balken

Hinweis: Ist "n" gleich 0, dann bezieht sich "n" auf den aktuellen Tag. Da maximale und minimale Werte nicht als n=0 festgelegt werden, erhalten wir ein Ergebnis für vorherige Tage, aber nicht für den aktuellen Tag.

Konstanten für die im Chart angezeigten Zeiteinheiten werden mit eckigen Klammern verwendet, wobei bei Konstanten bzgl. des Tagespreises runde Klammern benutzt werden.

```
Close[3]
```
➡ Der Schlusskurs vor 3 Perioden

```
Dclose(3)
```
➡ Der Schlusskurs vor 3 Tagen

Zeitliche Konstanten

Die Zeit ist ein oft vernachlässigter Bestandteil der technischen Analyse. Allerdings wissen Trader nur zu gut, wie wichtig gewisse Zeitspannen des Tages oder Zeitpunkte eines Jahres sind. Sie können in Ihren Programmen Zeit und Datum berücksichtigen, um die Effizienz Ihrer Indikatoren zu verbessern. Die zeitlichen Konstanten sind hier folgend beschrieben:

- **Date:** Diese Konstante zeigt das Datum des Schlusskurses jedes einzelnen Balkens im Format JahrMonatTag an (YYYYMMDD)

Die zeitlichen Konstanten werden von ProBuilder als ganze Zahl betrachtet. Die Konstante "Date" muss z.B. als eine aus 8 Zeichen zusammengesetzte Zahl verwendet werden.

Lassen Sie uns die Codezeile schreiben:

```
RETURN Date
```

Angenommen heute ist der 4. Juli 2020. Die oben erwähnte Codezeile gibt den Wert 20200704 an.

Das Datum wird auf folgende Weise gelesen:

20200704 = 2020 Jahre 07 Monate und 04 Tage.

Beachten Sie dabei, dass MM nie größer als 12 und JJ nie größer als 31 sein kann.

- **Time:** Diese Konstante zeigt Uhrzeit des Schlusskurses jedes Balkens im Format StundenMinutenSekunden (HHMMSS) an.

Beispiel:

```
RETURN Time
```

Dieser Indikator zeigt den Schlusskurs jedes Balkens im Format HHMMSS:

Es ist auch möglich, **Time** und **Date** zum Erstellen von Analysen oder zur Anzeige von Ergebnissen zu einem bestimmten Zeitpunkt in demselben Indikator zu verwenden. Im folgenden Beispiel wollen wir unseren Indikator auf das Datum des 1. Oktober und auf 9 Uhr 00 und 1 Sekunde vormittags exakt begrenzen:

```
a = (Date = 20081001)
b = (Time = 090001)
RETURN (a AND b)
```

Die folgenden Konstanten funktionieren auf gleiche Weise:

- **Minute:** Minute des Schlusskurses von jedem Balken (von 0 bis 59); nur bei Intraday-Charts.
- **Hour:** Stunde des Schlusskurses von jedem Balken (von 0 bis 23); nur bei Intraday-Charts.
- **Day:** Tag des Schlusskurses von jedem Balken (vom 1 bis 28 bzw. 29, 30 oder 31).
- **Month:** Monat des Schlusskurses von jedem Balken (vom 1 bis 12).
- **Year:** Jahr des Schlusskurses von jedem Balken.
- **DayOfWeek:** Wochentag des Schlusskurses von jedem Balken (keine Wochenendtage) (1=Montag, 2=Dienstag, 3=Mittwoch, 4=Donnerstag, 5=Freitag)

Beispiel:

```
a = (Hour > 17)
b = (Day = 30)
RETURN (a AND b)
```

- **CurrentHour:** aktuelle Stunde (vom lokalen Markt)
- **CurrentMinute:** aktuelle Minute (vom lokalen Markt)
- **CurrentMonth:** aktueller Monat (vom lokalen Markt)
- **CurrentSecond:** aktuelle Sekunde (vom lokalen Markt)
- **Today:** heutiges Datum (vom lokalen Markt)
- **CurrentTime:** aktuelle StundeMinuteSekunden (vom lokalen Markt)
- **CurrentYear:** aktuelles Jahr (vom lokalen Markt)
- **CurrentDayOfWeek:** aktueller Wochentag mit der Zeitzone des Marktes als Bezugspunkt

Der Unterschied zwischen den oben erwähnten "Current"-Konstanten und den "Nicht-Current"-Konstanten liegt im Aspekt des Aktuellen.

Das folgende Bild zeigt den Unterschied (angewandt auf die **CurrentTime** und **Time**-Anweisungen). Wir können die Tatsache hervorheben, dass bei "aktuellen" Befehlen die Zeitachse ignoriert werden muss und nur der angezeigte Wert berücksichtigt werden darf (der Wert der aktuellen Uhrzeit wird über den vollständigen Verlauf des Charts dargestellt).

Time zeigt die Schlusszeit von jedem Balken an.

CurrentTime zeigt die aktuelle Uhrzeit des Marktes an.

Wenn Sie Ihre Indikatoren mit Zählern (Anzahl der vorherigen Tage, Anzahl der vorherigen Balken, usw.…) versehen möchten, dann können Sie die Konstanten Days, BarIndex und IntradayBarIndex verwenden.

- **Days :** Zählt die Tage seit 1970

Diese Anweisung ist recht hilfreich, wenn Sie die Anzahl der vorherigen Tage kennen möchten. Dies ist besonders wichtig, wenn Sie mit (x) Tick oder (x) Volumen-Ansichten arbeiten.

Das folgende Beispiel zeigt Ihnen die Anzahl der Tage seit 1970.

```
RETURN Days
```

- **BarIndex:** Zählt die Anzahl an Balken seit Beginn der historischen Daten

Der Zähler beginnt von links nach rechts und zählt jeden Balken, einschließlich des aktuellen Balkens. Der zuerst geladene Balken hat die Balken-Nummer 0. Der Befehl BarIndex wird meistens zusammen mit IF-Anweisungen verwendet, die später im Handbuch vorgestellt werden.

- **IntradayBarIndex:** Zählt die Anzahl an Intraday-Balken

Der Zähler zeigt die Anzahl der Balken seit Markteröffnung und wird zu Beginn jedes neuen Tages auf Null zurückgesetzt. Der zuerst geladene Balken hat die Balken-Nummer 0.

Lassen Sie uns beide Zähler-Konstanten mithilfe von zwei voneinander getrennten Indikatoren vergleichen:

```
RETURN BarIndex
```

und

```
RETURN IntradayBarIndex
```

Wir können den Unterschied klar erkennen: IntradayBarIndex wird zu Beginn jedes neuen Tages auf Null zurückgesetzt.

Vom Preis abgeleitete Konstanten

Diese Konstanten ermöglichen es Ihnen, im Vergleich zu Open, High, Low und Close umfangreichere Informationen zu erhalten, da sie diese Kurselemente miteinander kombinieren, um gewisse Aspekte der Finanzmarktpsychologie beim aktuellen Balken hervorzuheben.

- **Range:** Spanne zwischen Hoch und Tief.
- **TypicalPrice:** Durchschnitt von Hoch-, Tief- und Schlusskurs.
- **WeightedClose:** Gewichteter Durchschnitt von Hoch (Gewichtung 1), Tief (Gewichtung 1) und Schlusskurs (Gewichtung 2)
- **MedianPrice:** Durchschnitt zwischen Hoch und Tief.
- **TotalPrice:** Durchschnitt zwischen Eröffnung, Hoch,Tief und Schluss.

Range zeigt die Volatilität (Preisschwankung) des gegenwärtigen Balkens an, ein Indikator in Bezug auf die Nervosität der Investoren.

Sie können einen Indikator mithilfe einer der oben erwähnten Konstanten z.B. durch die Programmierung einer einzigen Linie wie "RETURN Range" erstellen oder die Konstanten zum Programmieren komplexerer Indikatoren verwenden.

WeightedClose legt einen besonderen Wert auf den Schlusskurs des Preis-Balkens (noch wichtiger bei Anwendung auf Tages- oder Wochenbalken).

TypicalPrice und **TotalPrice** betonen die Geldmarktpsychologie im Verlauf des Tages, da sie 3 oder 4 vorherrschende Preise beim gegenwärtigen Balken berücksichtigen (siehe oben).

MedianPrice verwendet das Median-Konzept (Mittelwert) anstelle des Durchschnittskonzepts, welches sehr nützlich bei der Erstellung theoretischer Modelle sein kann, die die Anlegerpsychologie nicht berücksichtigen.

Range in % :

```
MyRange = Range
Calcul = (MyRange / MyRange[1] - 1) * 100
RETURN Calcul
```

Die unbestimmte Konstante

Das Stichwort **Undefined** ermöglicht es Ihnen, den Wert des Indikators in der Workstation nicht anzeigen zu lassen.

- **Undefined:** unbestimmte Daten (gleichwertig mit einem leeren Eingabefeld)

Ein Anwendungsbeispiel wird weiter hinten im Handbuch vorgestellt.

Verwenden von bereits vorhandenen Indikatoren

Bis jetzt haben wir Ihnen die Möglichkeiten von ProBuilder in Bezug auf Konstanten beschrieben und wie die Werte von vorherigen Balken mithilfe dieser Konstanten in Indikatoren einbezogen werden können. Bereits in der Workstation ProRealTime vorhandene Indikatoren funktionieren auf gleiche Weise sowie auch die von Ihnen programmierten Indikatoren.

ProBuilder-Indikatoren bestehen aus drei Bestandteilen mit folgender Syntax:

NameFunktion [berechnet über n Perioden] (angewendet auf Preis oder Indikator)

Wenn Sie die Schaltfläche "Funktion Einfügen" verwenden, um nach einer ProBuilder-Funktion zu suchen, und diese dann entsprechend Ihrem Programmcode hinzufügen, dann werden sowohl für Periode als auch Preis bzw. Indikator-Argument Standardwerte vorgegeben.

Average[20](Close)

Die Werte können verändert werden: z.B. können wir die 20 standardmäßig festgelegten Balken durch jedwede andere Balkenanzahl ersetzen (z.B. Average[10], Average[15], Average[30], ..., Average[n]). Genauso können wir "Close" durch "Open" oder "RSI (Relative Strength Index) ersetzen. Dies würde folgendes Beispiel ergeben:

Average[20](RSI[5])

Hier nun einige Programmbeispiele:

Das Programm berechnet den exponentiellen Gleitenden Durchschnitt über 20 Perioden, angewandt auf den Schlusskurs:

```
RETURN ExponentialAverage[20](Close)
```

Das Programm berechnet den gewichteten Gleitenden Durchschnitt über 20 Balken, angewandt auf den typischen Preis

```
RETURN WeightedAverage[20](TypicalPrice)
```

Das Programm berechnet den Wilder Durchschnitt über 100 Balken, angewandt auf das Volumen

```
RETURN WilderAverage[100](Volume)
```

Der folgende Programmcode berechnet den MACD (als Histogramm) angewandt auf den Schlusskurs. Der MACD basiert auf der Differenz zwischen dem exponentiellen Gleitenden Durchschnitt (EMA) über 12 Perioden minus EMA über 26 Perioden. Die MACD-Linie wird dann mithilfe eines exponentiellen Gleitenden Durchschnitts über 9 Perioden geglättet, um die Signallinie zu erhalten. MACD ist daher die Differenz zwischen MACD-Linie und Signallinie.

```
// Berechnung der MACD-Linie
MACDLine = ExponentialAverage[12](Close) - ExponentialAverage[26](Close)
// Berechnung der MACD Signal Linie
MACDSignalLine = ExponentialAverage[9](MACDLine)
// Berechnung der Differenz zwischen MACD-Linie und dessen Signal
MACDHistogramme = MACDLine - MACDSignalLine
RETURN MACDHistogramme
```

Variablen

Bei der Programmierung Ihres Indikators möchten Sie vielleicht Variablen einarbeiten. Die Variablen-Option in der oberen rechten Ecke im Fenster ermöglicht es Ihnen, einem unbestimmten Wert einen Standardwert zuzuordnen und diesen im "Einstellungsfenster" des Indikators ohne Veränderung des Programmcodes zu verwalten.

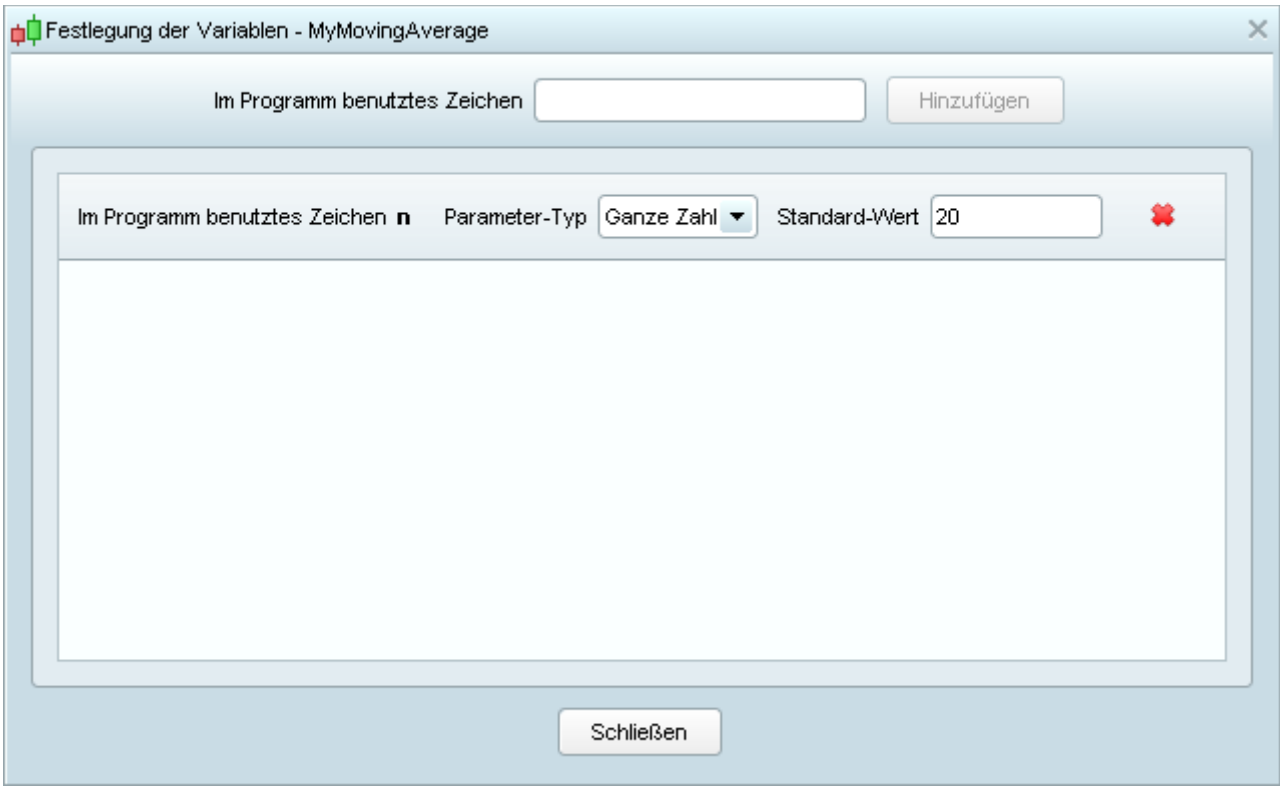
Lassen Sie uns einen einfachen Gleitenden Durchschnitt über 20 Perioden berechnen:

```
RETURN Average[20](Close)
```

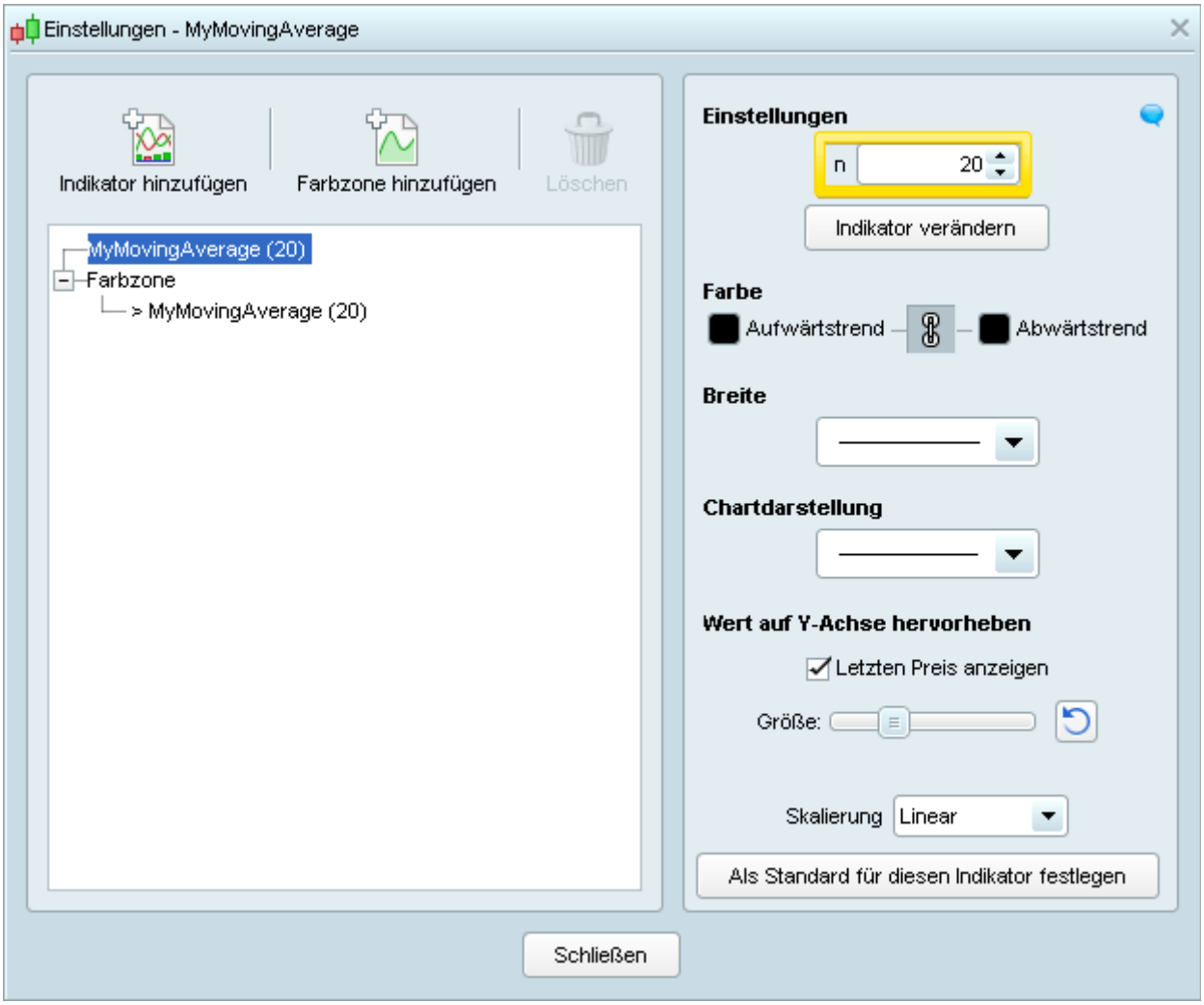
Zur Änderung der zur Berechnung festgelegten Periodenanzahl innerhalb des Einstellungsfensters des Indikators ersetzen Sie 20 durch die Variable "n":

```
RETURN Average[n](Close)
```

Klicken Sie dann auf "Hinzufügen". Dann klicken Sie auf "Hinzufügen" in "Variable" und ein neues Fenster "Definition von Variable" öffnet sich. Füllen Sie es folgendermaßen aus:



Klicken Sie auf die "OK" Taste. Danach werden Sie im Einstellungsfenster (in diesem Fall "Einstellung MyMovingAverage") einen neuen Parameter erhalten, der es Ihnen ermöglicht, die Anzahl der Perioden bei der Berechnung des Gleitenden Durchschnitts zu verändern:



Natürlich ist es möglich, das Gleiche mit vielen anderen Variablen zu tun, was Ihnen wiederum erlaubt, mehrere Variablen bei demselben Indikator gleichzeitig zu beeinflussen.

Kapitel II: Math. Funktionen und ProBuilder-Anweisungen

Kontrollstrukturen

Bedingte IF-Anweisung

Der **IF**-Befehl wird zur Erstellung eines bedingungsabhängigen Ablaufes verwendet, d.h. diese Aktion soll ausgeführt werden, wenn eine oder mehrere Bedingungen eintreffen.

Die Struktur wird durch die Anweisungen **IF**, **THEN**, **ELSE**, **ELSIF**, **ENDIF** gebildet, die abhängig von der Vielschichtigkeit der definierten Bedingungen verwendet werden. Wir werden diese der Reihe nach vorstellen.

Eine Bedingung, ein Ergebnis (IF THEN ENDIF)

Wir können nach einer Bedingung suchen und eine Aktion bestimmen, wenn die Bedingung zutreffend (wahr) ist. Sollte allerdings die Bedingung ungültig sein, dann wird nichts geschehen (standardmäßig, Result = 0).

In diesem Beispiel wird festgelegt, dass, wenn sich der aktuelle Kurs oberhalb des Gleitenden Durchschnittes über 20 Perioden befindet, dann folgendes im Chart angezeigt wird: Result = 1.

```
Result=0                                  RESULT ist gleich 0
IF Close > Average[20](Close) THEN        WENN Schlusskurs > GD über 20-Perioden
   Result = 1                             DANN Result = 1, andernfalls Result =0
ENDIF                                     ENDE DER BEDINGUNG
RETURN Result
```

RETURN muss immer zusammen mit einer Variable verwendet werden, damit das Ergebnis im Chart erscheint (im oberen Beispiel verwenden wir die Variable "Result").

Eine Bedingung, zwei Ergebnisse (IF THEN ELSE ENDIF)

Wir können auch ein anderes Ergebnis festlegen, wenn die Bedingung nicht zutrifft. Lassen Sie uns zum oberen Beispiel zurückkehren: Wenn der Kurs oberhalb des Gleitenden Durchschnittes über 20 Perioden verläuft, dann Anzeige von 1, andernfalls Anzeige von -1.

```
IF Close > Average[20](Close) THEN
   Result = 1
ELSE
   Result = -1
ENDIF
RETURN Result
```

NB: Wir haben hier einen binären Indikator erstellt. Für weitere Informationen, lesen Sie den Abschnitt in Bezug auf das Erstellen von binären oder ternären Indikatoren im hinteren teil dieses Handbuchs.

Sequentielle IF-Bedingungen

Nach der Bestätigung der Hauptbedingung können wir Unter-Bedingungen erstellen, d.h. die Bedingungen müssen nacheinander für gültig erklärt werden. Dafür müssen Sie eine Reihe (Sequenz) von **IF**-Strukturen in der entsprechenden Reihenfolge erstellen. Sie sollten darauf achten, genauso viele **ENDIF**-Anweisungen wie **IF**-Anweisungen festzulegen. Beispiel:

Doppel-Bedingungen bei Gleitenden Durchschnitten:

```
IF (Average[12](Close) - Average[20](Close) > 0) THEN
   IF ExponentialAverage[12](Close) - ExponentialAverage[20](Close) > 0 THEN
      Result = 1
   ELSE
      Result = -1
   ENDIF
ENDIF
RETURN Result
```

Mehrfach-Bedingungen (IF THEN ELSE ELSIF ENDIF)

Sie können für eine bestimmte Bedingung ein bestimmtes Ergebnis festlegen. Der Indikator signalisiert also mehrere Zustände: Wenn Bedingung 1 eintrifft, dann führe Aktion 1 aus; andernfalls, wenn Bedingung 2 eintrifft, dann führe Aktion 2 aus, … Wenn keine der bisher genannten Bedingungen eintrifft, dann führe Aktion n aus.

Diese Struktur verwendet folgende Bedingungen: **IF**, **THEN**, **ELSIF**, **THEN**.... **ELSE**, **ENDIF**.

Die Syntax lautet:

```
IF (Condition1) THEN
   (Action1)
ELSIF (Condition2) THEN
   (Action2)
ELSIF (Condition3) THEN
   (Action3)
...
...
...
ELSE
   (Action n)
ENDIF
```

Sie können **ELSIF** auch durch **ELSE IF** ersetzen, jedoch werden Sie beim Schreiben des Programms mehr Zeit benötigen. Sie müssen allerdings noch die Schleife beenden und dabei genauso viele **ENDIFs** wie zuvor festgelegte **IFs** hinzufügen. Wenn Sie mehrere Bedingungen in Ihrem Programm benutzen möchten, empfehlen wir Ihnen die Verwendung von **ELSIF** anstelle von **ELSE IF**.

Beispiel: Erkennen von bearischem und bullischem Engulfing

Dieser Indikator zeigt den Wert 1 an, wenn ein bullisches Engulfing erkennt wird, den Wert -1 bei einem bearischen Engulfing und 0 in der übrigen Zeit.

```
// Beschreibung des bullischen Engulfing
Condition1 = Close[1] < Open[1]
Condition2 = Open < Close[1]
Condition3 = Close > Open[1]
Condition4 = Open < Close

// Beschreibung des bearischen Engulfing
Condition5 = Close[1] > Open[1]
Condition6 = Close < Open
Condition7 = Open > Close[1]
Condition8 = Close < Open[1]

IF Condition1 AND Condition2 AND Condition3 AND Condition4 THEN
   a = 1
ELSIF Condition5 AND Condition6 AND Condition7 AND Condition8 THEN
   a = -1
ELSE
   a = 0
ENDIF
RETURN a
```

Beispiel: Demarks Widerstandspivots

```
IF DClose(1) > DOpen(1) THEN
   Phigh = DHigh(1) + (DClose(1) - DLow(1)) / 2
   Plow = (DClose(1) + DLow(1)) / 2
ELSIF DClose(1) < DOpen(1) THEN
   Phigh = (DHigh(1) + DClose(1)) / 2
   Plow = DLow(1) - (DHigh(1) - DClose(1)) / 2
ELSE
   Phigh = DClose(1) + (DHigh(1) - DLow(1)) / 2
   Plow = DClose(1) - (DHigh(1) - DLow(1)) / 2
ENDIF
RETURN Phigh , Plow
```

Beispiel: BarIndex

Im Kapitel 1 unseres Handbuchs haben Ihnen den **BarIndex** als einen Zähler der geladenen historischen Balken vorgestellt. **BarIndex** wird häufig zusammen mit **IF** verwendet. Wenn wir z.B. wissen möchten, ob die Anzahl der Balken in unserem Diagramm mehr oder weniger als 23 Balken beträgt, dann schreiben wir:

```
IF BarIndex <= 23 THEN
   a = 0
ELSIF BarIndex > 23 THEN
   a = 1
ENDIF
RETURN a
```

Iterative Schleife FOR

Die Schleife **FOR** wird verwendet, wenn eine endliche Reihe von Elementen überprüft werden soll. Die Reihe muss aus ganzen Zahlen bestehen (z.B. 1,2,3,...,6,7 oder 7,6,…,3,2,1).

Die Struktur setzt sich aus den Komponenten **FOR**, **TO**, **DOWNTO**, **DO**, **NEXT** zusammen. **TO** und **DOWNTO** werden abhängig von der Reihenfolge des Auftretens aufsteigend oder absteigend verwendet. Hierbei ist zu beachten, dass alle sich zwischen **FOR** und **DO** befindenden Elemente die Extremitäten des zu überprüfenden Intervalls darstellen.

Ansteigend (FOR, TO, DO, NEXT)

```
FOR (Variable = BeginningValueOfTheSeries) TO EndingValueOfTheSeries DO
   (Action)
NEXT
```

Beispiel: Glätten des Gleitenden Durchschnitts über 12 Perioden (GD12)

Lassen Sie uns eine Variable erstellen (Result), welche nacheinander die Perioden 11, 12 und 13 des Gleitenden Durchschnitts summiert.

```
Result = 0
FOR Variable = 11 TO 13 DO
   Result = Average[Variable](Close) + Result
NEXT
REM Lassen Sie uns eine Variable erstellen (AverageResult), die den Durchschnitt der 3 Gleitenden Durchschnitte errechnet, indem sie "Result" durch 3 teilt. "AverageResult" führt dadurch eine Glättung durch.
AverageResult = Result / 3
RETURN AverageResult
```

Lassen Sie uns die Berechnungen Schritt für Schritt nachvollziehen:
Mathematisch gesehen möchten wir den Durchschnitt vom Gleitenden Durchschnitt über 11, 12 und 13 Perioden berechnen.
Variable erhält dann nacheinander den Wert 11, 12 und 13 (FOR arbeitet immer mit ganzen Zahlen).
Result = 0
Variable = 11
Result erhält den Wert des vorherigen Result + GD11 = (0) + GD11 = (0 + GD11)
Wir gehen zum nächsten Wert des Zählers.
Variable = 12
Result erhält den Wert des vorherigen Result + GD12 = (0 + GD11) + GD12 = (0 + GD11 + GD12)
Wir gehen zum nächsten Wert des Zählers.
Variable = 13
Der Zähler empfängt den folgenden Wert.
La valeur 13 est la dernière valeur du compteur.
Result erhält den Wert des vorherigen Result + GD13 = (0 + GD11 + GD12) + GD13 = (0 + GD11 + GD12 + GD13)
Der Wert 13 ist der letzte Wert des Zählers.
Die "**FOR**"-Schleife wird mit der Anweisung "**NEXT**" geschlossen.
Dann wird AverageResult angezeigt.
Dieser Code bedeutet ganz einfach, dass "Variable" zuerst den Anfangswert der Reihe übernimmt und dann den folgenden Wert (Vorheriger + 1) empfängt und so weiter bis zum allerletzten Wert der Reihe. Abschließend wird die Schleife geschlossen.
Beispiel: Durchschnitt des höchsten Kurses über die letzten 20 Balken

```
SUMhigh = 0
IF BarIndex < 5 THEN                Wenn noch keine 5 Perioden vorhanden sind,
   MMhigh = Undefined               dann teilen wir "MAhigh" standardmäßig den Wert "unbestimmt" zu
ELSE                                Ansonsten
   FOR i = 0 TO 4 DO                Für Werte zwischen 0 und 4
      SUMhigh = High[i]+SUMhigh     summieren wir die letzten 5 "High" Werte
   NEXT
ENDIF
MMhigh = SUMhigh / 5                Wir teilen diese Summe durch 5 und ordnen diesen Wert "MAhigh" zu
RETURN MMhigh                       Anzeige von MAhigh
```

Absteigende Folge (FOR, DOWNTO, DO, NEXT)

Zum Festlegen einer absteigenden Folge werden die folgenden Anweisungen verwendet: **FOR**, **DOWNTO**, **DO**, **NEXT**.
Die Syntax lautet:

```
FOR (Variable = WertEndederSerie) DOWNTO WertBeginnderSerie DO
   (Action)
NEXT
```

Lassen Sie uns zum oberen Beispiel bezüglich des Gleitenden Durchschnitts der Hochs über 5 Perioden zurückkehren: Wir können feststellen, dass lediglich die Extremitäten des überprüften Zeitraums abgewandelt wurden.

```
SUMhigh = 0
IF BarIndex < 5 THEN
   MMhigh = Undefined
ELSE
   FOR i = 4 DOWNTO 0 DO
      SUMhigh = High[i] + SUMhigh
   NEXT
ENDIF
MMhigh = SUMhigh / 5
RETURN Mmhigh
```

Bedingte Schleife WHILE

WHILE wird verwendet, um eine Aktion auszuführen, solange die Bedingung zutrifft. Sie werden sehen, dass diese Anweisung große Ähnlichkeit mit den einfachen Befehlen **IF**/**THEN**/**ENDIF** hat.

Diese Struktur verwendet die folgenden Befehle: **WHILE**, **DO** (beliebig), **WEND**

Die Syntax lautet:

```
WHILE (Condition) DO
    (Action 1)
    …
    (Action n)
WEND
```

Dieser Code erlaubt das Hervorheben der Anzahl an Balken, die zwischen dem aktuellen Balken und dem vorherigen höchsten Balken liegen, bei einer Höchstanzahl von 30 Balken.

Beispiel:

```
i = 1
WHILE high > high [i] and i < 30 DO
   i = i + 1
WEND
RETURN i
```

Beispiel: Der folgende Indikator berechnet die Anzahl von aufeinander folgenden steigenden Phasen.

```
Increase = (Close > Close[1])
Count = 0
WHILE Increase[Count] DO
   Count = Count + 1
WEND
RETURN Count
```

*Allgemeine Anmerkung zur bedingten Anweisung **WHILE**:*

*Ähnlich wie bei der **IF**-Anweisung wird das Programm die bedingte Schleife nicht berücksichtigen, solange die Bedingung unbestätigt bleibt.*

Zum Beispiel:

```
Count = 0
WHILE i <> 11 DO
   i = i + 1
   Count = Count + 1
WEND
RETURN Count
```

Im oberen Code-Beispiel wird die Variable i nicht definiert. Der **WHILE**-Befehl erkennt daher nicht den Wert von i. Folglich kann nicht getestet werden, ob i gleich 10 ist oder nicht. Die Variable "Count" wird nicht berücksichtigt werden. Demzufolge gibt der Zähler den Wert 0 aus.

Der korrekte Code lautet:

```
i = 0
Count = 0
WHILE i <> 11 DO
   i = i + 1
   Count = Count + 1
WEND
RETURN Count
```

In diesem Code ist die Variable i korrekt festgelegt. Die Schleife arbeitet nun richtig, da deren Anfangsbedingung nun gültig ist.

BREAK

Die Anweisung **BREAK** erlaubt gewissermaßen einen Zwangsausstieg aus einer **WHILE**- oder **FOR**-Schleife. Kombinationen sind mit der **IF**-Anweisung möglich, sowohl innerhalb einer **WHILE**- als auch einer **FOR**-Schleife.

Mit WHILE

Wenn wir eine bedingte **WHILE**-Schleife beenden möchten, ohne dabei auf eine bestimmte Situation zu warten, in der die Bedingung gültig ist, verwenden wir **BREAK** auf folgende Weise:

```
WHILE (Condition) DO
    (Action)
    IF (ConditionBreak)
        BREAK
    WEND
```

Die Verwendung von **BREAK** in einer **WHILE**-Schleife ist nur dann von Interesse, wenn wir eine zusätzliche Bedingung überprüfen möchten, deren Wert nur innerhalb der **WHILE**-Schleife erkannt wird. Nehmen wir als Beispiel einen auf einem Oszillator basierenden Stochastik-Indikator, der nur im Fall eines ansteigenden Trends berechnet wird:

```
line = 0
Increase = (Close - Close[1]) > 0
i = 0
WHILE Increase[i] DO
       i = i + 1
       // Wenn high - low, Ausstieg aus der Schleife zum Vermeiden einer Division durch Null.
       IF (high-low) = 0 then
          BREAK
       ENDIF
       osc = (close - low) / (high - low)
       line = AVERAGE [i] (osc)
WEND
RETURN line
```

Mit FOR

Wenn wir aus einer iterativen **FOR**-Schleife aussteigen möchten, ohne den letzten (oder ersten) Wert der Reihe zu erreichen, dann verwenden wir **BREAK** mit folgender Syntax.

```
FOR (Variable = WertEndederSerie) TO WertBeginnderSerie DO
    (Action)
    BREAK
NEXT
```

Nehmen wir als Beispiel einen Indikator, der die aufeinander folgenden ansteigenden Volumina über die letzten 19 Perioden kumuliert. Dieser Indikator gibt den Wert 0 aus, wenn das Volumen fällt.

```
Indicator = 0
FOR i = 0 TO 19 DO
    IF (Volume[i] > Volume[i + 1]) THEN
       Indicator = Indicator + 1
    ELSE
       BREAK
    ENDIF
NEXT
RETURN Indicator
```

Wäre der Befehl "**BREAK**" in diesem Code nicht vorhanden, dann würde sich die Schleife bis 19 (letztes Element der Serie) fortsetzen, auch wenn die Volumen-Bedingung nicht erfüllt würde.

Allerdings bewirkt der Befehl **BREAK** das Ergebnis 0, sobald die Bedingung nicht mehr erfüllt ist.

CONTINUE

Mit der Anweisung **CONTINUE** können Sie das Programm in der Anfangszeile einer **WHILE**- oder **FOR**-Schleife fortsetzen, und das ohne Neustart der kompletten Schleife (jeder Zähler wird allerdings erhöht und der Wert nicht auf 0 zurückgesetzt). Dieser Befehl wird häufig zusammen mit **BREAK** verwendet, entweder zum Ausstieg aus einer Schleife (**BREAK**) oder zum Verbleiben in der Schleife (**CONTINUE**).

Mit WHILE

Lassen Sie uns ein Programm erstellen, welches die Anzahl derjenigen Candlesticks summiert, deren Schlusskurs oberhalb und deren Eröffnungskurs unterhalb des vorherigen Candlesticks liegen. Wird die Bedingung nicht erfüllt, wird der Zähler auf 0 zurückgesetzt.

```
Increase = Close > Close[1]
Count = 0
WHILE Open < Open[1] DO
   IF Increase[Count] THEN
      Count = Count + 1
      CONTINUE
   ENDIF
BREAK
WEND
RETURN Count
```

Mithilfe von **CONTINUE** wird die **WHILE**-Schleife nicht beendet, wenn die **IF**-Bedingung nicht erfüllt wird. Dies ermöglicht uns, die Anzahl an Mustern zu zählen, welche die Bedingung erfüllen. Ohne die **CONTINUE**-Anweisung würde das Programm die Schleife beenden, auch wenn die **IF**-Bedingung erfüllt wurde. Dann wären wir nicht mehr in der Lage, die erkannten Muster weiterhin zu zählen und wir hätten entsprechend ein binäres Ergebnis (1, 0).

Mit FOR

Lassen Sie uns ein Programm erstellen, welches die Anzahl der Candlesticks zählt, deren Schlusskurs über demjenigen des vorherigen Candlesticks liegt. Wenn die Bedingung nicht erfüllt ist, wird der Zähler auf 0 zurückgesetzt.

```
Increase = Close > Close[1]
Count = 0
FOR i = 1 TO BarIndex DO
   IF Increase[Count] THEN
      Count = Count + 1
      CONTINUE
   ENDIF
BREAK
NEXT
RETURN Count
```

FOR ermöglicht die Überprüfung der Bedingung auf der Basis aller zur Verfügung stehenden historischen Daten. Mithilfe von **CONTINUE** wird die **FOR**-Schleife nicht beendet, sollte die IF-Bedingung nicht erfüllt werden, sondern mit dem nächsten Wert i fortgesetzt. Dies ermöglicht uns, die Anzahl an Mustern zu zählen, die die Bedingung erfüllen.

Ohne die **CONTINUE**-Anweisung würde das Programm die Schleife beenden, auch wenn die **IF**-Bedingung erfüllt wurde. Dann wären wir nicht mehr in der Lage, die erkannten Muster weiterhin zu zählen und wir hätten entsprechend ein binäres Ergebnis (1, 0).

ONCE

Die **ONCE**-Anweisung ("**NUR EINMAL**") wird verwendet, um einer Variable einen bestimmten Wert einmalig zuzuweisen.

Mit dem Wissen, dass die Programmiersprache bei jedem Programm vor der Anzeige des Ergebnisses den Code so oft liest wie historische Balken angezeigt werden, muss daher berücksichtigt werden, dass **ONCE**

- vom Programm nur ein einziges Mal berücksichtigt wird (einschließlich einer erneuten Überprüfung),
- während der zweiten Überprüfung die auf der Basis der ersten Überprüfung ermittelten Werte beibehält.

Zum besseren Verständnis dieser Anweisung muss die Vorgehensweise verdeutlicht werden, in der die Programmsprache den Code liest; daher das folgende Beispiel.

Dies sind zwei Programme, die die Ergebnisse 0 und 15 ausgeben und deren einziger Unterschied darin besteht, dass der **ONCE**-Befehl beim zweiten Programm hinzugefügt wurde:

Programm 1

```
1 Count = 0
2 i = 0
3 IF i <= 5 THEN
4    Count = Count + i
5    i = i + 1
6 ENDIF
7 RETURN Count
```

Programm 2

```
1 ONCE Count = 0
2 ONCE i = 0
3 IF i <= 5 THEN
4    Count = Count + i
5    i = i + 1
6 ENDIF
7 RETURN Count
```

Sehen wir uns an, wie die Programmiersprache die Codes liest.

Programm 1 :

Die Programmiersprache wird zuerst die Zeile 1 (L1: Count = 0; i = 0) lesen, anschließend L2, L3, L4, L5 und L6 (Count = 0; i = 1) und dann ab Zeile 1 das Ganze genau auf dieselbe Weise noch einmal lesen. Aufgrund von **RETURN** wird aus dem Programm ausgestiegen, sobald letzteres "n Mal" gelesen wurde. Für den nächsten Balken beginnt das Programm am Anfang und sowohl i als auch Count werden auf 0 gesetzt, wodurch als Ergebnis immer 0 ausgegeben wird (wie am Ende der ersten Lektüre).

Programm 2 :

Die Programmiersprache wird die Zeile 1 (L1: Count = 0; i = 0) lesen, anschließend L2, L3, L4, L5 und L6 (Count = 0; i = 1); sobald es bei der **RETURN**-Zeile ankommt, fängt es bei der Schleife in Zeile 3 an (**die beiden Zeilen mit ONCE werden nur beim ersten Mal berücksichtigt**), dann Zeilen 4, 5, 6 (Count = 1 ; i = 2); dann geht es wieder zurück (Count = 3; i = 3) und so weiter zu (Count = 15; i = 6). Bei diesem Ergebnis angekommen, wird die **IF**-Schleife nicht mehr berücksichtigt, da die Bedingung nicht mehr erfüllt wird; die einzige, zum Lesen bleibende Zeile ist L7, somit wird als Ergebnis für die übrigen geladenen Balken der Wert 15 ausgegeben.

Mathematische Funktionen

Übliche unäre und binäre Operatoren

Lassen Sie uns jetzt die mathematischen Funktionen näher betrachten. In ProBuilder stehen die wichtigsten in der Mathematik bekannten Funktionen zur Verfügung. Bitte beachten Sie, dass a und b Beispiele sind, die durch Zahlen oder andere Variablen ersetzt werden können.

- **MIN**(a, b): berechnet das Minimum von a und b
- **MAX**(a, b): berechnet das Maximum von a und b
- **ROUND**(a): rundet zur nächstliegenden geraden Zahl auf oder ab
- **ABS**(a): berechnet den absoluten Wert von a
- **SGN**(a): zeigt das Zeichen a (1, wenn positiv, -1, wenn negativ)
- **SQUARE**(a): berechnet das Quadrat
- **SQRT**(a): berechnet die Quadratwurzel von a
- **LOG**(a): berechnet den natürlichen Logarithmus (Neperianer) von a
- **EXP**(a): berechnet den Exponentialfaktor von a
- **COS**(a): berechnet den Kosinus von a
- **SIN**(a): berechnet den Sinus von a
- **TAN**(a): berechnet den Tangens von a
- **ATAN**(a): berechnet den Arkustangens von a

Lassen Sie uns z.B. die mathematische Normalverteilung programmieren. Dies ist interessant, weil hier Quadratfunktion, Quadratwurzelfunktion und die exponentielle Funktion gleichzeitig verwendet werden:

```
REM Normalverteilung auf x=10 angewendet, Standardabweichung = 6 und MathExpectation = 8
REM Festlegen als optimierte Variablen:
StandardDeviation = 6
MathExpectation = 8
x = 10
Indicator = EXP((1 / 2) * (SQUARE(x - MathExpectation) / StandardDeviation)) /
(StandardDeviation * SQRT(2 / 3.14))
RETURN Indicator
```

Übliche mathematische Operatoren

- **a < b:** a ist kleiner als b
- **a <= b oder a =< b:** a ist kleiner oder gleich b
- **a > b:** a ist größer als b
- **a >= b oder a => b:** a ist größer oder gleich b
- **a = b:** a ist gleich b (oder b wird a zugeordnet)
- **a <> b:** a ist verschieden zu b

Chart-Vergleichsfunktionen

- **a CROSSES OVER b:** die a-Kurve überkreuzt die b-Kurve steigend
- **a CROSSES UNDER b:** die a-Kurve unterkreuzt die b-Kurve fallend

Summenfunktion

- **cumsum:** Berechnet die Summe eines Kurses oder Indikators über alle im Chart geladenen Balken

Die Syntax ist:

```
cumsum (Kurs oder Indikator)
```

Bsp.: `cumsum (Close)` berechnet die Summe der Schlusskurse aller im Chart geladenen Balken.

- **summation:** Berechnet die Summe eines Kurses oder Indikators über eine festzulegende Anzahl an Balken

Die Summe errechnet sich ab dem letzten Wert (von rechts nach links)

Die Syntax von **Summation** ist:

```
summation[Anzahl an Balken](Kurs oder Indikator)
```

Bsp.: `summation[20](Open)` berechnet die Summe der Eröffnungskurse der letzten 20 Balken.

Statistische Funktionen

Die Schreibweise all dieser Funktionen folgt dem gleichen Prinzip wie jene bei Indikatoren oder der Summenfunktion, d.h.:

```
lowest[Anzahl an Balken](Kurs oder Indikator)
```

- **lowest:** zeigt den niedrigsten Wert über die festgelegte Periodenanzahl an
- **highest:** zeigt den höchsten Wert über die festgelegte Periodenanzahl an
- **STD:** zeigt die Standardabweichung bei einem Wert über die festgelegte Periodenanzahl an
- **STE:** zeigt den Standardfehler bei einem Wert über die festgelegte Periodenanzahl an

Logische Operatoren

Wie bei jeder Programmiersprache ist es notwendig, logische Operatoren zur Verfügung zu haben, um aussagekräftige Indikatoren erstellen zu können. Hier nun die 4 logischen Operatoren in ProBuilder:

- **NOT**(a): logisch NICHT
- a **OR** b: logisch ODER
- a **AND** b: logisch UND
- a **XOR** b: exklusiv ODER

Berechnung des Trend-Indikators: On Balance Volume (OBV)

```
IF NOT((Close > Close[1]) OR (Close = Close[1])) THEN
   MyOBV = MyOBV - Volume
ELSE
   MyOBV = MyOBV + Volume
ENDIF
RETURN MyOBV
```

ProBuilder-Anweisungen

- **RETURN:** zeigt das Ergebnis an
- **CustomClose:** zeigt einen anpassbaren Kurswert; standardmäßig den Schlusskurs "Close"
- **CALL:** bezieht eine zuvor vom Nutzer erstellte Funktion mit ein
- **AS:** benennt die angezeigten unterschiedlichen Ergebnisse
- **COLOURED:** färbt die angezeigte Kurve in der Farbe Ihrer Wahl

RETURN

Wir haben bereits im ersten Kapitel die Bedeutung der Anweisung **RETURN** gesehen. Zum Vermeiden von Programmierfehlern sollten einige spezifische Eigenschaften des Befehls bekannt sein.

Für eine richtige Verwendung in einem Programmcode sollte beachtet werden, dass **RETURN**

- nur ein einziges Mal im Programm vorkommen darf,
- immer in der letzten Zeile des Codes steht,
- optional mit anderen Anweisungen wie z.B. **AS** und **COLOURED** benutzt werden kann,
- zusammen mit der Ergebnisvariable verwendet werden kann, um viele Ergebnisse anzuzeigen; in diesem Fall werden die einzelnen Variablen durch ein Komma getrennt (Beispiel: **RETURN** a,b).

REM oder //

Durch die Anweisung **REM** bzw. die beiden Schrägstriche können dem Code Anmerkungen bzw. Kommentare hinzugefügt werden. Das ist sehr nützlich, um sich daran zu erinnern, was der Code tun soll. Diese Kommentare werden natürlich nicht vom Programm gelesen. Lassen Sie uns das Konzept am folgenden Beispiel zeigen:

```
// Dieses Programm berechnet den Gleitenden Durchschnitt über 20 Perioden, angewandt auf den Schlusskurs
RETURN Average[20](Close)
```


Verwenden Sie keine Sonderzeichen (Beispiele: ö, ä, ü, ß …) in ProBuilder (dieses gilt nicht für den **REM**-Bereich)

CustomClose

CustomClose ist eine Anweisung zur Anzeige von Konstanten wie **Close**, **Open**, **High**, **Low** und vielen anderen, welche im Einstellungsfenster des Indikators ausgewählt werden können.

Die Syntax ist so wie bei den anderen Konstanten und wird an die im Chart angezeigte Zeiteinheit angepasst:

```
CustomClose[n]
```

Nehmen wir ein einfaches Beispiel:

```
RETURN CustomClose[2]
```

Durch einen Klick auf den Schraubenschlüssel links oben im Chart sehen Sie im Einstellungsfenster, dass die zur Berechnung verwendeten Kurse angepasst werden können.

CALL

CALL erlaubt Ihnen einen selbst geschriebenen Indikator zu verwenden, der sich bereits in der Workstation befindet.

Die schnellste Methode zum Hinzufügen eines bestehenden Indikators erfolgt über die Kategorie "Nutzer-Indikatoren" nach einem Klicken auf "Funktionen einfügen".

Stellen wir uns vor, Sie haben den MACD-Indikator als Histogramm selber programmiert und unter dem Namen "HistoMACD" abgespeichert.

Wählen Sie Ihren Indikator in der Kategorie aus und klicken Sie auf "Hinzufügen". Sie sehen dann im Programmfenster:

```
myHistoMACD = CALL HistoMACD
```

Die Software hat Ihren ursprünglichen Indikator "HistoMACD" automatisch in "myHistoMACD" umbenannt.

Dies bedeutet, dass Sie jedes Mal, wenn Sie den Indikator "HistoMACD" in diesem Programmcode verwenden möchten, diesen immer über "myHistoMACD" einfügen müssen.

AS

Das Schlüsselwort **AS** erlaubt es, die verschiedenen angezeigten Ergebnisse zu benennen. Diese Anweisung wird zusammen mit **RETURN** auf folgende Weise benutzt:

```
RETURN Result1 AS "Curve Name", Result2 AS "Curve Name", …
```

Der Vorteil dieses Befehls liegt darin begründet, dass die einzelnen Bestandteile des Indikators visuell leichter voneinander unterschieden werden können.

Beispiel:

```
a = ExponentialAverage[200](Close)
b = WeightedAverage[200](Close)
c = Average[200](Close)
RETURN a AS "Exponential Average", b AS "Weighted Average", c AS "Arithmetical Average"
```

COLOURED

Die Anweisung **COLOURED** wird hinter **RETURN** verwendet, um die Linien oder Kurven unterschiedlich nach der RGB-Norm einzufärben (rot, grün, blau). Es können die Hauptfarben dieser Norm zugeordnet werden:

FARBE	RGB WERT (ROT, GRÜN, BLAU)	DEUTSCH
	(0, 0, 0)	Schwarz
	(255, 255, 255)	Weiß
	(255, 0, 0)	Rot
	(0, 255, 0)	Grün
	(0, 0, 255)	Blau
	(255, 255, 0)	Gelb
	(0, 255, 255)	Türkis
	(255, 0, 255)	Lila

Die Syntax der Coloured-Anweisung lautet:

```
RETURN Indicator COLOURED(Red, Green, Blue)
```

Die **AS**-Anweisung kann mit der **COLOURED** (.,.,.) kombiniert werden, wenn folgende Reihenfolge berücksichtigt wird:

```
RETURN Indicator COLOURED(Red, Green, Blue) AS "Name meiner Kurve"
```

Wir gehen noch einmal zum vorherigen Beispiel zurück und fügen der Zeile "**RETURN**" die Anweisung **COLOURED** hinzu.

```
a = ExponentialAverage[200](Close)
b = WeightedAverage[200](Close)
c = Average[200](Close)
RETURN a COLOURED(255, 0, 0) AS "Exponential Moving Average", b COLOURED(0, 255, 0) AS "WeightedMoving Average", c COLOURED(0, 0, 255) AS "Simple Moving Average"
```

Dieses Bild zeigt die individuelle farbliche Anpassung der Linien.

Progammieranweisungen zum Zeichnen

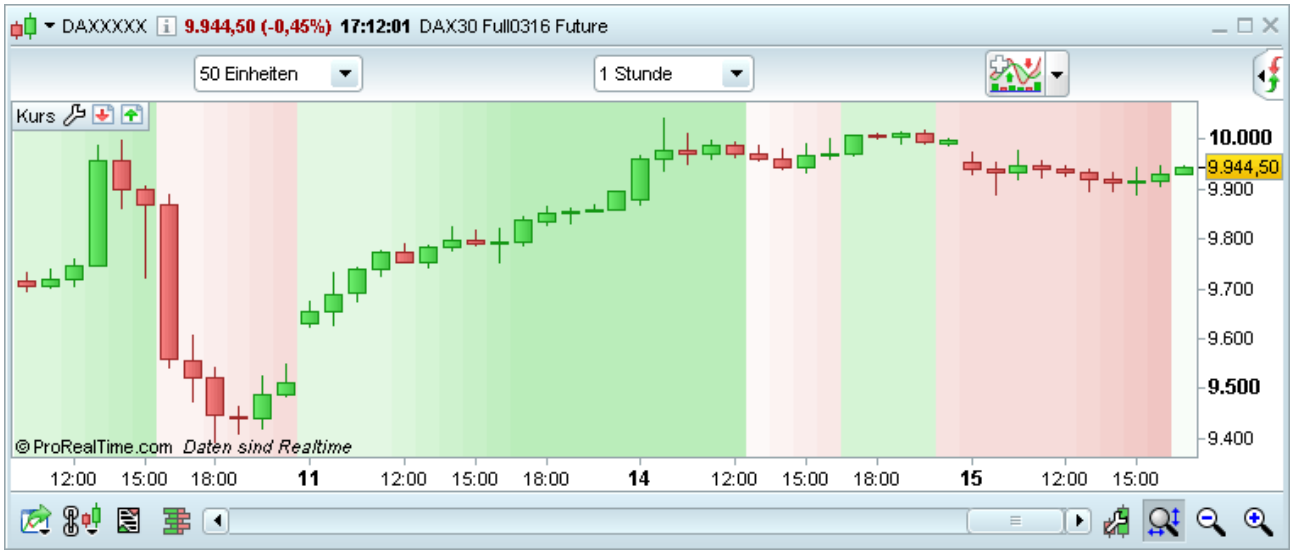
Die folgenden Anweisungen stehen ab der Version 10.3 der Software zur Verfügung. Hiermit können Sie Objekte in die Charts zeichnen, aber auch Ihre Kerzen oder Balken individuell anpassen.

- BACKGROUNDCOLOR: Hiermit kann der Hintergrund der Charts oder bestimmte Balken (wie z.B. gerade oder ungerade Tage) eingefärbt werden. Die Farbzone beginnt auf halbem Wege zwischen dem vorherigen und dem darauf folgenden Balken

  Beispiel: `BACKGROUNDCOLOR (0, 127, 255, 25)`

  Sie können auch eine Farbe durch eine Variable ersetzen, wenn Sie die Hintergrundfarbe aufgrund Ihrer Bedingungen verändern möchten.

  Beispiel: `BACKGROUNDCOLOR (0, color, 255, 25)`



- DRAWBARCHART: Zeichnet einen individuellen Balken in den Chart. Open, High, Low und Close können Konstanten oder Variablen sein.

  Beispiel: `DRAWBARCHART (open, high, low, close) COLOURED (0, 255, 0)`

- DRAWCANDLE: Zeichnet einen individuellen Candlestick in den Chart. Open, High, Low und Close können Konstanten oder Variablen sein.

  Beispiel: `DRAWCANDLE (open, high, low, close) COLOURED (0, 255, 0)`

- DRAWARROW: Zeichnet einen nach rechts zeigenden Pfeil. Sie müssen einen Punkt für diesen Pfeil festlegen (X und Y-Achse). Sie können auch eine Farbe auswählen.

Beispiel: `DRAWARROW (x1, y1) COLOURED (R, V, B, a)`

- DRAWARROWUP: Zeichnet einen nach oben zeigenden Pfeil. Sie müssen den Ausgangspunkt des Pfeils festlegen. Sie können auch eine Farbe auswählen.

Beispiel: `DRAWARROWUP (x1, y1) COLOURED (R, V, B, a)`

Dies ist zum Hinzufügen von visuellen Kaufsignalen sinnvoll.

- DRAWARROWDOWN: Zeichnet einen nach unten zeigenden Pfeil. Sie müssen einen Punkt für den Pfeil (X und Y-Achse) festlegen. Sie können auch eine Farbe auswählen.

Beispiel: `DRAWARROWDOWN (x1, y1) COLOURED (R, V, B, a)`

Dies ist zum Hinzufügen von visuellen Verkaufssignalen sinnvoll.

- DRAWRECTANGLE: Zeichnet ein Rechteck in den Chart.

Beispiel: `DRAWRECTANGLE (x1, y1, x2, y2) COLOURED (R, V, B, a)`

- DRAWELLIPSE : Zeichnet eine Ellipse in den Chart.

Beispiel: `DRAWELLIPSE (x1, y1, x2, y2) COLOURED (R, V, B, a)`

- DRAWLINE: Zeichnet eine Linie in den Chart.

Beispiel: `DRAWLINE (x1, y1, x2, y2) COLOURED (R, V, B, a)`

- DRAWHLINE : Zeichnet eine horizontale Linie in den Chart.

Beispiel: `DRAWHLINE (y1) COLOURED (R, V, B, a)`

- DRAWVLINE: Zeichnet eine vertikale Linie in den Chart.

Beispiel: `DRAWVLINE (x1) COLOURED (R, V, B, a)`

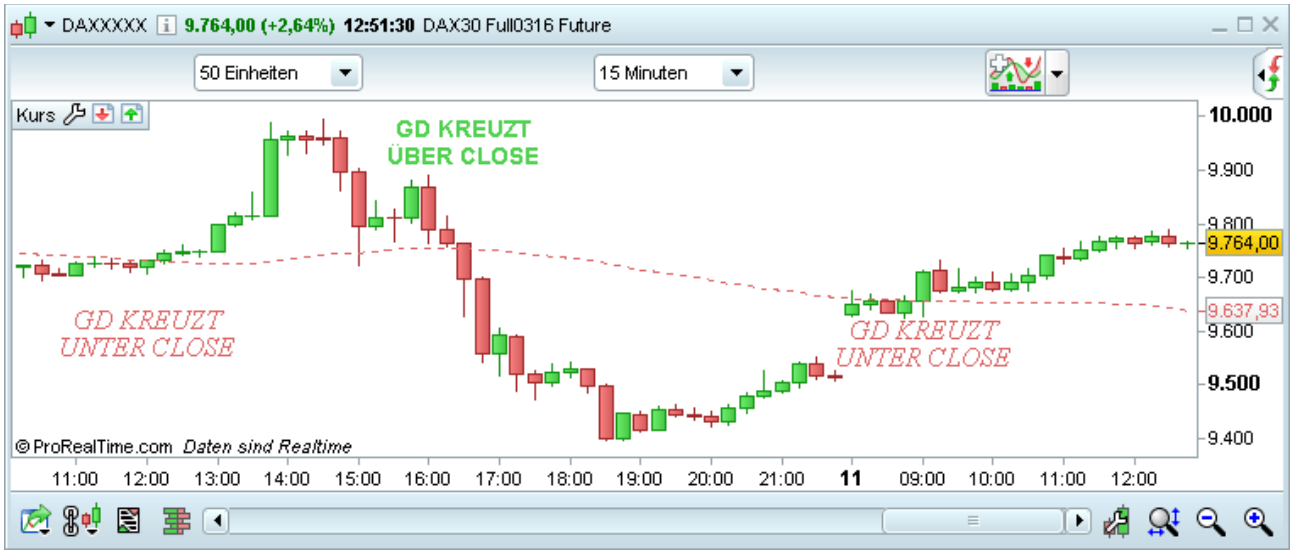
- DRAWSEGMENT: Zeichnet ein Segment in den Chart.

Beispiel: `DRAWSEGMENT (x1, y1, x2, y2) COLOURED (R, V, B, a)`

Beispiel: `DRAWSEGMENT (barindex, close, barindex[5], close[5])`

- DRAWTEXT: Hiermit wird an einem bestimmten Ort im Chart ein Textfeld Ihrer Wahl hinzugefügt.

Beispiel: `DRAWTEXT ("your text", x1, y1) COLOURED (R, V, B, a)`



- DRAWONLASTBARONLY: Mithilfe dieses Parameters können Sie Objekte nur auf den letzten Balken zeichnen. Dieser Parameter sollte zur Optimierung von Berechnungen immer zusammen mit "CALCULATEONLASTBARS" verwendet werden.

Beispiel: `DEFPARAM DRAWONLASTBARONLY = true`

- CALCULATEONLASTBARS: Dieser Parameter ermöglicht das Erhöhen der Geschwindigkeit, mit der Indikatoren berechnet werden. Dadurch wird festgelegt, dass in den Ergebnissen die jüngsten Balken zuerst angezeigt werden.

Beispiel: `DEFPARAM CALCULATEONLASTBARS = 200`

Kapitel III: Praktische Aspekte

Erstellen eines binären bzw. ternären Indikators: Wie und Warum?

Ein binärer bzw. ternärer Indikator ist ein Indikator, der lediglich zwei oder drei mögliche Ergebnisse ausgeben kann (normalerweise 0, 1 oder -1). Sein Hauptzweck bei Börsenanalysen besteht darin, bestimmte auf dem Indikator basierende Muster oder Bedingungen schnell im Chart identifizieren zu können.

Zweck des binären oder ternären Indikators:

- Erkennen der wichtigsten Candlestickmuster (z.B.: Harami, Morning Star, Hammer, …).
- Erleichtern der Chartanalyse, wenn man nach bestimmten Bedingungen sucht.
- Erstellen von auf einer Bedingung basierenden Alarmen, wenn der zugrunde Indikator gleichzeitig nach mehreren Bedingungen sucht ➡ Sie haben also mehr Alarmsignale zur Verfügung!
- Erkennen komplexer Bedingungen auch bei den historischen Daten.
- Einfachere Erstellung eines Backtest.

Binäre und ternäre Indikatoren sind im Wesentlichen IF-Strukturen. Wir empfehlen Ihnen eine erneute Lektüre des entsprechenden Abschnitts, bevor Sie weiter lesen.

Wir werden Ihnen jetzt einige Indikatoren zum Erkennen von Candlestick-Mustern vorstellen:

Binärer Indikator: Hammer

```
// Erkennen des Hammers
Hammer = Close>Open AND High = Close AND (Open-Low) >= 3*(Close-Open)
IF Hammer THEN
   Result = 1
ELSE
   Result = 0
ENDIF
RETURN Result AS "Hammer"
```

Ternärer Indikator: Golden Cross und Death Cross

```
a = ExponentialAverage[10](Close)
b = ExponentialAverage[20](Close)
c = 0
// Erkennen des Golden Cross
IF a CROSSES OVER b THEN
   c = 1
ENDIF
// Erkennen des Death Cross
IF a CROSSES UNDER b THEN
   c = -1
ENDIF
RETURN c
```

Hinweis: Es werden hier zwei exponentielle Gleitende Durchschnitte über 10 und 20 Perioden angezeigt, die auf den Schlusskurs angewendet werden, um die Übereinstimmung mit den Indikator-Signalen besser hervorzuheben.

Sie finden weitere Indikatoren zum Erkennen von Candlestick-Mustern weiter hinten in diesem Handbuch.

Erstellen von STOP-Indikatoren: Überwachen Sie Ihre Positionen in Echtzeit

Es ist möglich, Indikatoren auf der Basis persönlicher Einstellungen zu erstellen, die STOPs entsprechen und mögliche Ausstiegssignale erzeugen.

Mit dem ProBackTest-Modul zum Erstellen von Strategien, welches in einem gesonderten Handbuch behandelt wird, lassen sich auch Stop-Levels eines Handelssystems festlegen. Die Programmierung eines Stops als Indikator ist aus folgenden Gründen interessant:

- Der STOP kann als Linie im Chart dargestellt werden, die in Echtzeit im Chart aktualisiert wird.
- Es ist unnötig, Kauf-oder Verkauf-Orders festzulegen (anders als im Rahmen eines ProBackTests)
- Es können dem Indikator Alarme hinzugefügt werden, um beim Eintreffen der Bedingung unverzüglich informiert zu werden.

Die Programmierung von Stops erlaubt es Ihnen, die meisten der bisher vorgestellten Programmieranweisungen anzuwenden.

Es gibt 4 Kategorien von Stops, die wir nachfolgend vorstellen:

- **STOP mit statische Gewinnmitnahme**
- **STOP mit statischem Loss**
- **STOP bei Inaktivität**
- **Trailing STOP**

Die in den folgenden Beispielen dargestellten Indikatoren sind Programmierbeispiele für Stop-Indikatoren. Sie müssen diese auf der Basis der in den vorherigen Kapiteln erlernten Anweisungen Ihren eigenen Parametern anpassen.

STOP mit statischer Gewinnmitnahme

Ein *statischer Take-Profit* stellt ein Preislevel dar, bei dessen Erreichen die Position geschlossen wird. Dieser Stop stellt per se ein feststehendes Level dar (horizontale Linie). Der Nutzer dieses STOPs nimmt also seine Gewinne mit.

Der untere Indikator definiert zwei Levels zum Positionseinstieg zum Zeitpunkt "Start":

- Wenn Sie Käufer sind, dann berücksichtigen Sie die obere Kurve, die einen Gewinn von 10% darstellt, also 110% des Kurses zum Kaufzeitpunkt.
- Wenn Sie im Leerverkauf sind, dann berücksichtigen Sie die untere Kurve, die ebenfalls einen Gewinn von 10% darstellt, also 90% des Kurses zum Verkaufszeitpunkt.

Der Code sieht folgendermaßen aus.

Wir schlagen Ihnen hier ein STOP-Beispiel vor, das individuell angepasst werden kann:

```
// Wir legen als optimierte Variable fest:
// StartingTime = 100000
// Passen Sie diese Variable korrekt Ihrer Uhrzeit zum Einstieg in die Position an
// Price = Preis zum Zeitpunkt der Positionseinnahme (in diesem Beispiel wird die Position um 10 Uhr eingenommen)
// Bei einer Long-Position überwachen Sie die obere Linie, bei einer Short-Position die untere Linie.
// AmplitudeUp stellt die Veränderungsrate des Kurses dar, der zum Zeichnen des Take Profits bei Long-Positionen verwendet wird (standardmäßig: 1.1)
// AmplitudeDown stellt die Veränderungsrate des Kurses dar, der zum Zeichnen des Take Profits bei Short-Positionen verwendet wird (standardmäßig: 0.9)
IF Time = StartingTime THEN
   StopLONG = AmplitudeUp * Price
   StopSHORT = AmplitudeDown * Price
ENDIF
RETURN StopLONG COLOURED(0, 0, 0) AS "TakeProfit LONG", StopSHORT COLOURED(0, 255, 0) AS "TakeProfit SHORT"
```

Statischer STOP Loss

Ein *STOP Loss* ist das Gegenteil eines *Take-Profit STOPs*; dies bedeutet, dass wir den Positionsausstieg auf einem niedrigeren Kurslevel anstelle auf einem höheren Preislevel positionieren. Dieser STOP ist sinnvoll, um Verluste auf ein Minimum zu begrenzen.

Wie beim Take-Profit wird auch dieser STOP einem festliegenden Niveau zugeordnet.

Der untere Indikator definiert zwei Levels zum Positionseinstieg zum Zeitpunkt "Start":

- Wenn Sie Käufer sind, dann berücksichtigen Sie die untere Kurve, die einen Verlust von 10% darstellt, also 90% des Kurses zum Kaufzeitpunkt.
- Wenn Sie im Leerverkauf sind, dann berücksichtigen Sie die obere Kurve, die ebenfalls einen Verlust von 10% darstellt, also 110% des Kurses zum Verkaufszeitpunkt.

Der Code sieht folgendermaßen aus:

```
// Wir legen als optimierte Variable fest:
// StartingTime = 100000 (in diesem Beispiel wird die Position um 10 Uhr eingenommen)
// Passen Sie diese Variable korrekt Ihrer Uhrzeit zum Einstieg in die Position an
// Price = Preis zum Zeitpunkt der Positionseinnahme
// AmplitudeUp stellt die Veränderungsrate des Kurses dar, der zum Zeichnen des Stop Loss bei Long-Positionen verwendet wird (standardmäßig: 0.9)
// AmplitudeDown stellt die Veränderungsrate des Kurses dar, der zum Zeichnen des Stop Loss bei Short-Positionen verwendet wird (standardmäßig: 1.1)
IF Time = StartingTime THEN
   StopLONG = AmplitudeUp * Price
   StopSHORT = AmplitudeDown * Price
ENDIF
RETURN StopLONG COLOURED(0, 0, 0) AS "StopLoss LONG", StopSHORT COLOURED(0, 255, 0) AS "StopLoss SHORT"
```

Inaktivitäts-STOP

Ein *Inaktivitäts-STOP* schließt die Position, wenn das Gewinnziel (in % oder Punkten) über einen bestimmten Zeitraum (Anzahl an Balken) nicht erreicht wurde.

Beispiel eines Inaktivitäts-Stops in einem Intraday-Chart:

Dieser Stopp muss zusammen mit zwei Indikatoren verwendet werden:

- Der Erste liegt über der Preiskurve
- Der Zweite sollte separat angezeigt werden

Indikator1

```
// MyVolatility = 0.01 entspricht der Veränderungsrate zwischen oberen und unterem Band der festgelegten Spanne
IF IntradayBarIndex = 0 THEN
   ShortTarget = (1 - MyVolatility) * Close
   LongTarget = (1 + MyVolatility) * Close
ENDIF
RETURN ShortTarget AS "ShortTarget", LongTarget AS "LongTarget"
```

Indikator2

```
// Wir definieren als optimierte Variable:
// Einstieg in Position zum Marktpreis
// MyVolatility = 0.01 entspricht der Veränderungsrate zwischen oberen und unterem Band der festgelegten Spanne
// NumberOfBars = 20 entspricht dem maximalen Zeitraum (in Balken), über den sich der Preis entwickeln kann, ohne dass die Positionen glattgestellt werden (Ergebnis bei 1)
Result = 0
Cpt = 0
IF IntradayBarIndex = 0 THEN
   ShortTarget = (1 - MyVolatility) * Close
   LongTarget = (1 + MyVolatility) * Close
ENDIF
FOR i = IntradayBarIndex DOWNTO 1 DO
   IF Close[i] >= ShortTarget AND Close[i] <= LongTarget THEN
      Cpt = Cpt + 1
   ELSE
      Cpt = 0
   ENDIF
   IF Cpt = NumberOfBars THEN
      Result = 1
   ENDIF
NEXT
RETURN Result
```

Trailing STOP

Ein *Trailing STOP* folgt dynamisch der Kursentwicklung und zeigt an, zu welchem Zeitpunkt die Position zu schließen ist.

Wir zeigen Ihnen unten zwei Variationen des Trailing Stops: Der Erste entspricht einem dynamischen Trailing Stop-Loss und der Zweite einem dynamischen Take Profit.

Dynamischer Trailing STOP-LOSS (zum Verwenden im Intraday-Handel)

```
// Wir definieren als optimierte Variable:
// StartingTime = 090000 (in diesem Beispiel wird die Position um 9 Uhr eingenommen; passen Sie diese Variable korrekt Ihrer Uhrzeit zum Einstieg in die Position an)
// Einstieg in Position zum Marktpreis
// Amplitude stellt die Veränderungsrate der Kurven "Cut" in Bezug auf die Kurven "Lowest" dar (es kann z.B. die Amplitude = 0.95 verwendet werden)
IF Time = StartingTime THEN
   IF lowest[5](Close) < 1.2 * Low THEN
      IF lowest[5](Close) >= Close THEN
         Cut = Amplitude * lowest[5](Close)
      ELSE
         Cut = Amplitude * lowest[20](Close)
      ENDIF
   ELSE
      Cut = Amplitude * lowest[20](Close)
   ENDIF
ENDIF
RETURN Cut AS "Trailing Stop Loss"
```

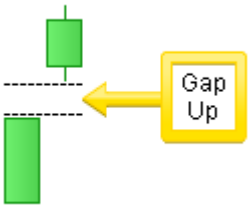
Dynamischer Trailing Take-Profit-STOP (zum Verwenden im Intraday-Handel)

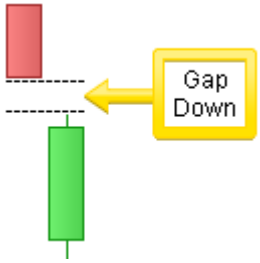
```
// Wir definieren als optimierte Variable:
// StartingTime = 090000 (in diesem Beispiel wird die Position um 9 Uhr eingenommen; passen Sie diese Variable korrekt Ihrer Uhrzeit zum Einstieg in die Position an)
// Einstieg in Position zum Marktpreis
// Amplitude stellt die Veränderungsrate der Kurven "Cut" in Bezug auf die Kurven "Lowest" dar (es kann z.B. die Amplitude = 1.015 verwendet werden)
IF Time = StartingTime THEN
   StartingPrice = Close
ENDIF
Price = StartingPrice - AverageTrueRange[10]
TrailingStop = Amplitude * highest[15](Price)
RETURN TrailingStop COLOURED (255, 0, 0) AS "Trailing take profit"
```

Kapitel IV: Übungen

Candlestick-Muster

- **Aufwärts-GAP oder Abwärts-Gap**





Die Farbe der Kerzen hat keinerlei Bedeutung.

Es wird als optimierte Variable eine Amplitude von 0.001 festgelegt.

Ein Gap entsteht aufgrund von zwei Bedingungen:

- (die aktuelle Eröffnung des Tages liegt strikt oberhalb des Schlusskurses des Vortages) oder (die aktuelle Eröffnung des Tages liegt strikt unterhalb des Schlusskurses des Vortages)
- der absolute Wert von ((aktueller Eröffnung des Tages - Schlusskurs des Vortages) / Schlusskurs des Vortages) ist absolut größer als die Amplitude

```
// Initialisierung der Amplitude (des Gaps)
Amplitude = 0.001
// Initialisierung des Signalmelders
Detector = 0
// Gap Up
// Erste Existenzbedingung eines Gaps
IF Low > High[1] THEN
   // Zweite Existenzbedingung eines Gaps
   IF ABS((Low - High[1]) / High[1]) > Amplitude THEN
      // Verhalten des Signalmelders
      Detector = 1
   ENDIF
ENDIF
// Gap Down
// Erste Existenzbedingung eines Gaps
IF High < Low[1] THEN
   // Zweite Existenzbedingung eines Gaps
   IF ABS((High - Low[1]) / Low[1]) > Amplitude THEN
      // Verhalten des Signalmelders
      Detector = -1
   ENDIF
ENDIF
// Anzeige des Ergebnisses
RETURN Detector AS "Gap detection"
```

- **Doji (flexible Version)**

In diesem Code legen wir den Doji als Candlestick fest, dessen Range (Hoch - Schluss) 5 Mal höher ist als der absolute Wert von (Eröffnung - Schluss).

```
Doji = Range > ABS(Open - Close) * 5
RETURN Doji AS "Doji"
```

- **Doji (strikte Version)**

Bei diesem Doji stimmen Schlusskurs und Eröffnungskurs überein.

```
Doji = (Open = Close)
RETURN Doji AS "Doji"
```

Indikatoren

- **BODY MOMENTUM**

Der Body Momentum wird mathematisch durch folgende Formel berechnet:

BodyMomentum = 100*BodyUp / (BodyUp + BodyDown)

BodyUp (wie auch BodyDown) zählt diejenigen Balken, deren Schlusskurs den Eröffnungskurs übersteigt (bzw. unterschreitet) und dies über einen festgelegten Zeitraum (hier z.B. 14 Perioden).

```
Periods = 14
b = Close - Open
IF BarIndex > Periods THEN
   Bup = 0
   Bdn = 0
   FOR i = 1 TO Periods
      IF b[i] > 0 THEN
         Bup = Bup + 1
      ELSIF b[i] < 0 THEN
         Bdn = Bdn + 1
      ENDIF
   NEXT
   BM = (Bup / (Bup + Bdn)) * 100
ELSE
   BM = Undefined
ENDIF
RETURN BM AS "Body Momentum"
```

ELLIOT WAVE OSCILLATOR

Der Elliot-Wave-Oszillator misst den Unterschied zwischen zwei Gleitenden Durchschnitten.

Der kurze Gleitende Durchschnitt stellt die Preisbewegung dar, während der lange Gleitende Durchschnitt die Grundtendenz aufzeigt.

Wenn die Preise eine Welle 3 bilden, dann steigen die Kurse stark an, was zu einem hohen Wert des Oszillators führt.

In einer Welle 5 steigen die Preise langsamer an und der Oszillator weist einen deutlich niedrigeren Wert auf.

```
RETURN Average[5](MedianPrice) - Average[35](MedianPrice) AS "Elliot Wave Oscillator"
```

Williams %R

Dieser Indikator ähnelt sehr dem Stochastik Oszillator. Zuerst müssen zwei Kurven festgelegt werden:

1) Die Kurve von Hochs zu Hochs über 14 Perioden

2) Die Kurve von Tiefs zu Tiefs über 14 Perioden

Die %R-Linie wird mit folgender Formel berechnet: (Close – LowestLow) / (HighestHigh –LowestLow) * 100

```
HighestH = highest[14](High)
LowestL = lowest[14](Low)
MyWilliams = (Close - LowestL) / (HighestH - LowestL) * 100
RETURN MyWilliams AS "Williams %R"
```

Bollinger Bänder

Diese Bänder basieren auf der Einrahmung eines einfachen Gleitenden Durchschnitts über 20 Perioden, angewandt auf den Schlusskurs.

Dieser Gleitende Durchschnitt wird oben (bzw. unten) durch eine Kurve auf der Basis von + (bzw. -) zweimal die Standardabweichung, angewandt auf die 20 vorherigen Schlusskurse, eingerahmt.

```
a = Average[20](Close)
// Festlegen der Standardabweichung
StdDeviation = STD[20](Close)
Bsup = a + 2 * StdDeviation
Binf = a - 2 * StdDeviation
RETURN a AS "Average", Bsup AS "Bollinger Up", Binf AS "Bollinger Down"
```

Glossar

A

CODE	SYNTAX	FUNKTION
ABS	ABS(a)	Mathematische Funktion "Absoluter Wert"
AccumDistr	AccumDistr(price)	Klassische Verteilungskumulierung
ADX	ADX[N]	Indikator 'Average Directional Index' oder 'ADX' über n Perioden
ADXR	ADXR[N]	Indikator 'Average Directional Index Rate' oder 'ADXR' über n Perioden
AND	a AND b	Logischer Operator UND
AroonDown	AroonDown[P]	Aroon Down über n Perioden
AroonUp	AroonUp[P]	Aroon Up über n Perioden
ATAN	ATAN(a)	Mathematische Funktion "Arkustangens"
AS	RETURN x AS "ResultName"	Anweisung zum Benennen einer Linie im Chart. Benutzt mit "RETURN".
Average	Average[N](price)	Einfacher Gleitender Durchschnitt über n Perioden
AverageTrueRange	AverageTrueRange[N](price)	"Average True Range" - geglättet nach der Wilder Methode

B

CODE	SYNTAX	FUNKTION
BACKGROUNDCOLOR	BACKGROUNDCOLOR(R,V,B,a)	Erlaubt das Einfärben des Chart-Hintergrunds oder bestimmter Balken (wie bei geraden bzw. ungeraden Tagen)
BarIndex	BarIndex	Zählt die im Chart angezeigten Candlesticks
BollingerBandWidth	BollingerBandWidth[N](price)	Bollinger Bandbreite Indikator
BollingerDown	BollingerDown[N](price)	Unteres Bollinger-Band (Unterstützung)
BollingerUp	BollingerUp[N](price)	Oberes Bollinger-Band (Widerstand)
BREAK	(FOR...DO...BREAK...NEXT) oder (WHILE...DO...BREAK...WEND)	Instruktion zum Abbrechen einer FOR- oder WHILE-Schleife

C

CODE	SYNTAX	FUNKTION
CALCULATEONLASTBARS	DEFPARAM CalculateOnLastBars = 200	Erlaubt das Erhöhen der Berechnungsgeschwindigkeit eines Indikators durch Festlegen der für das Ergebnis verwendeten Balkenanzahl
CALL	myResult = CALL myFunction	Einbeziehung eines persönlichen Indikators
CCI	CCI[N](price) oder CCI[N]	'Commodity Channel Index'
ChaikinOsc	ChaikinOsc[Ch1, Ch2](price)	'Chaikin-Oszillator'
Chandle	Chandle[N](price)	'Chande Momentum Oszillator'
ChandeKrollStopUp	ChandeKrollStopUp[Pp, Qq, X]	Schützender Stop nach Chande und Kroll bei Kauf-Positionen
ChandeKrollStopDown	ChandeKrollStopDown[Pp, Qq, X]	Schützender Stop nach Chande und Kroll bei Verkauf-Positionen
Close	Close[N]	Schlusskurs der aktuellen oder der n-letzten Kerze
COLOURED	RETURN x COLOURED(R,G,B)	Festlegen einer Chartlinie in einer Farbe nach RGB (nur Indikatoren)
COS	COS(a)	Kosinus-Funktion
CROSSES OVER	a CROSSES OVER b	Boolean-Operator zur Überprüfung, ob eine Kurve eine andere überkreuzt
CROSSES UNDER	a CROSSES UNDER b	Boolean-Operator zur Überprüfung, ob eine Kurve eine andere unterkreuzt
cumsum	cumsum(price)	Errechnet einen bestimmten Kurs anhand aller geladenen Daten
CurrentDayOfWeek	CurrentDayOfWeek	Aktueller Tag der Woche
CurrentHour	CurrentHour	Aktuelle Stunde
CurrentMinute	CurrentMinute	Aktuelle Minute
CurrentMonth	CurrentMonth	Aktueller Monat
CurrentSecond	CurrentSecond	Aktuelle Sekunde
CurrentTime	CurrentTime	Aktuelle Uhrzeit (HHMMSS)
CurrentYear	CurrentYear	Aktuelles Jahr
CustomClose	CustomClose[N]	Im Einstellungsfenster des Charts anpassbare Bedingung (Standard: Schlusskurs)
Cycle	Cycle(price)	Indikator 'Cycle'

D

CODE	SYNTAX	FUNKTION
Date	Date[N]	Datum, an dem der aktuelle Balken geschlossen wurde
Day	Day[N]	Tag, an dem der aktuelle Balken geschlossen wurde
Days	Days[N]	Zählt Tage seit 1900
DayOfWeek	DayOfWeek[N]	Wochentag, an dem der aktuelle Balken geschlossen wurde
DClose	DClose(N)	Schlusskurs vom n-ten Tag vor dem aktuellen Balken
DEMA	DEMA[N](price)	Doppelter exponentieller Gleitender Durchschnitt
DHigh	DHigh(N)	Tageshoch des n-ten Balkens vor dem aktuellen Balken
DI	DI[N](price)	Indikator 'Demand Index'
DIminus	DIminus[N](price)	Indikator DI
DIplus	DIplus[N](price)	Indikator DI+
DLow	DLow(N)	Tagestief des n-ten Tages vor dem aktuellen Balken
DO	Vgl. FOR und WHILE	Optionaler Befehl für 'FOR' und 'WHILE' zur Definition der Schleife
DOpen	DOpen(N)	Tageseröffnungskurs des n-ten Tages vor dem aktuellen Balken
DOWNTO	Vgl. FOR	Befehl in der 'FOR' Schleife zur Ausführung in absteigender Reihenfolge
DPO	DPO[N](price)	Indikator 'Detrended Price Oszillator'
DRAWARROW	DRAWARROW(x1,y1)	Zeichnet einen nach rechts zeigenden Pfeil
DRAWARROWDOWN	DRAWARROWDOWN(x1,y1)	Zeichnet einen nach unten zeigenden Pfeil
DRAWARROWUP	DRAWARROWUP(x1,y1)	Zeichnet einen nach oben zeigenden Pfeil
DRAWBARCHART	DRAWBARCHART(open,high,low,close)	Zeichnet einen persönlichen Barchart in den Chart. Open, high, low und close können Konstanten oder Variablen sein.
DRAWCANDLE	DRAWCANDLE(open,high,low,close)	Zeichnet einen persönlichen Candlestick in den Chart. Open, high, low und close können Konstanten oder Variablen sein.
DRAWELLIPSE	DRAWELLIPSE(x1,y1,x2,y2)	Zeichnet eine Ellipse in den Chart
DRAWHLINE	DRAWHLINE(y1)	Zeichnet eine horizontale Linie in den Chart
DRAWLINE	DRAWLINE(x1,y1,x2,y2)	Zeichnet eine Linie in den Chart
DRAWONLASTBARONLY	DEFPARAM DrawOnLastBarOnly = true	Dieser Parameter erlaubt es, Objekte nur auf dem letzten Balken zu zeichnen.

DRAWRECTANGLE	DRAWRECTANGLE(x1,y1,x2, y2)	Zeichnet ein Rechteck in den Chart
DRAWSEGMENT	DRAWSEGMENT(x1,y1,x2,y2)	Zeichnet ein Segment in den Chart
DRAWTEXT	DRAWTEXT("your text", x1, y1)	Hiermit kann dem Chart an einem bestimmten Ort ein Textfeld mit dem Inhalt Ihrer Wahl hinzugefügt werden
DRAWVLINE	DRAWVLINE(x1)	Zeichnet eine vertikale Linie in den Chart

E

CODE	SYNTAX	FUNKTION
EaseOfMovement	EaseOfMovement[l]	Indikator 'Ease of Movement'
ELSE	Vgl. IF/THEN/ELSE/ENDIF	Befehl zum Aufrufen der zweiten Bedingung innerhalb von IF-bedingten Bedingungen
ELSIF	Vgl. IF/THEN/ELSE/ENDIF	Steht für ELSE IF (wird innerhalb der bedingten Schleife verwendet)
EMV	EMV[N]	Indikator 'Ease of Movement Value'
ENDIF	Vgl. IF/THEN/ELSE/ENDIF	Schlussbefehl für IF-bedingte Anweisungen
EndPointAverage	EndPointAverage[N](price)	Endpunkt des Gleitenden Durchschnitts
EXP	EXP(a)	Mathematische Funktion "Exponentiell"
ExponentialAverage	ExponentialAverage[N](price)	Exponentieller Gleitender Durchschnitt

F - G

CODE	SYNTAX	FUNKTION
FOR/TO/NEXT	FOR i =a TO b DO a NEXT	FOR-Schleife (berechnet alle Werte in ansteigender (TO) oder absteigender (DOWNTO) Reihenfolge)
ForceIndex	ForceIndex(price)	Indikator 'Force Index' zum Ermitteln, wer zurzeit den Markt dominiert (Käufer oder Verkäufer)

H

CODE	SYNTAX	FUNKTION
High	High[N]	Hoch des aktuellen Balkens oder des n-letzten Balkens
highest	highest[N](price)	Höchster Kurs über eine vorher definierte Anzahl von Balken
HistoricVolatility	HistoricVolatility[N](price)	Historische oder statistische Volatilität
Hour	Hour[N]	Gibt die Stunde des Schlusskurses von jedem im Chart geladenen Balken an

I - J - K

CODE	SYNTAX	FUNKTION
IF/THEN/ENDIF	IF a THEN b ENDIF	Gruppe von bedingten Anweisungen ohne eine zweite Anweisung
IF/THEN/ELSE/ENDIF	IF a THEN b ELSE c ENDIF	Gruppe von bedingten Anweisungen
IntradayBarIndex	IntradayBarIndex[N]	Zählt die Anzahl an Balken in einem Intraday-Chart

L

CODE	SYNTAX	FUNKTION
LinearRegression	LinearRegression[N](price)	Indikator 'Lineare Regression'
LinearRegressionSlope	LinearRegressionSlope[N](price)	Steigung des Indikators 'Lineare Regression'
LOG	LOG(a)	Mathematische Funktion des natürlichen Logarithmus (Neperianer)
Low	Low[N]	Tief des aktuellen Balkens oder von dem n-letzten Balken
lowest	lowest[N](price)	Tiefstkurs über eine vorher definierte Anzahl von Balken

M

CODE	SYNTAX	FUNKTION
MACD	MACD[S,L,Si](price)	Indikator 'Moving Average Convergence Divergence (MACD)'
MACDline	MACDLine[S,L,Si](price)	Linie des MACD
MassIndex	MassIndex[N]	Indikator 'Mass Index' angewandt über n Balken
MAX	MAX(a,b)	Mathematische Funktion 'Maximum'
MedianPrice	MedianPrice	Durchschnitt von Hoch und Tief
MIN	MIN(a,b)	Mathematische Funktion 'Minimum'
Minute	Minute	Gibt die Minute von jedem im Chart geladenen Balken an
MOD	a MOD b	Mathematische Funktion 'Division mit Rest'
Momentum	Momentum[l]	Momentum-Indikator (Schluss – Schluss des n-letzten Balkens)
MoneyFlow	MoneyFlow[N](price)	Indikator 'MoneyFlow' (Ergebnis zwischen -1 und 1)
MoneyFlowIndex	MoneyFlowIndex[N]	Indikator 'MoneyFlow Index'
Month	Month[N]	Gibt den Monat von jedem im Chart geladenen Balken an

N

CODE	SYNTAX	FUNKTION
NEXT	Vgl. FOR/TO/NEXT	Schlussbefehl für die 'FOR'-Schleife
NOT	NOT a	Logischer Operator NICHT

O

CODE	SYNTAX	FUNKTION
OBV	OBV(price)	Indikator 'On-Balance-Volume'
ONCE	ONCE VariableName = VariableValue	Einführung einer Definitionsanweisung, die nur einmal ausgeführt wird
Open	Open[N]	Bezeichnet den Eröffnungspreis des aktuellen Balkens oder des n-letzten Balkens
OR	a OR b	Logischer Operator ODER

P - Q

CODE	SYNTAX	FUNKTION
Pipsize	Pipsize	Größe eines Pips (oder Punktes), PipSize=PointSize
PositiveVolumeIndex	PositiveVolumeIndex(price)	Indikator 'Positive Volume Index'
PriceOscillator	PriceOscillator[S,L](price)	Indikator 'Percentage Price oscillator'
PVT	PVT(price)	Indikator 'Price Volume Trend'

R

CODE	SYNTAX	FUNKTION
R2	R2[N](price)	Koeffizient R Quadrat (Fehlerquote der linearen Regression zum Kurs)
Range	Range[N]	Preisspanne zwischen Hoch und Tief des aktuellen Balkens
REM	REM comment	Kommentar im Code
Repulse	Repulse[N](price)	Repulse'-Indikator (misst die Stärke von Aufwärts- und Abwärtstrend bei jeder Kerze
RETURN	RETURN Result	Anweisung zur Anzeige des Ergebnisses (nur Indikatoren)
ROC	ROC[N](price)	Indikator 'Price Rate of Change'
ROUND	ROUND(a)	Mathematische Funktion "Runden zur nächsten Einheit"
RSI	RSI[N](price)	Oszillator 'Relative Strength Index'

S

CODE	SYNTAX	FUNKTION
SAR	SAR[At,St,Lim]	Indikator 'Parabolic SAR'
SARatdmf	SARatdmf[At,St,Lim](price)	Indikator 'Parabolic SAR ATDMF'
SIN	SIN(a)	Mathematische Funktion 'Sinus'
SGN	SGN(a)	Mathematische Funktion 'Zeichen von' (positiv oder negativ)
SMI	SMI[N,SS,DS](price)	Indikator 'Stochastik Momentum Index'
SmoothedStochastic	SmoothedStochastic[N,K](price)	Indikator 'Geglätteter Stochastik'
SQUARE	SQUARE(a)	Mathematische Funktion 'zum Quadrat'
SQRT	SQRT(a)	Mathematische Funktion 'Quadratwurzel'
STD	STD[N](price)	Statistische Funktion 'Standardabweichung
STE	STE[N](price)	Statistische Funktion 'Standardfehler'
Stochastic	Stochastic[N,K](price)	%K-Linie des Stochastik-Indikators
summation	summation[N](price)	Summe eines bestimmten Preises über die n-letzten Kerzen
Supertrend	Supertrend[STF,N]	Indikator 'SuperTrend'

T

CODE	SYNTAX	FUNKTION
TAN	TAN(a)	Mathematische Funktion 'Tangens'
TEMA	TEMA[N](price)	Dreifacher exponentieller Gleitender Durchschnitt
THEN	Vgl. IF/THEN/ELSE/ENDIF	Anweisung folgt der ersten Bedingung von 'IF'
Ticksize	Ticksize	Gibt die Tickgröße des Instrumentes an (die kleinstmögliche Variation des Preises)
Time	Time[N]	Zeigt die Uhrzeit im Format HHMMSS
TimeSeriesAverage	TimeSeriesAverage[N](price)	Gleitender Durchschnitt Zeitliche Serie
TO	Vgl. FOR/TO/NEXT	Richtungsanweisung in der "FOR"-Schleife
Today	Today[N]	Aktueller Tag im Datumsformat YYYYMMDD
TotalPrice	TotalPrice[N]	(Schluss + Eröffnung + Hoch + Tief) / 4
TR	TR(price)	Indikator 'True Range'
TriangularAverage	TriangularAverage[N](price)	Dreieckiger Gleitender Durchschnitt
TRIX	TRIX[N](price)	Dreifacher exponentieller Gleitender Durchschnitt
TypicalPrice	TypicalPrice[N]	Typischer Preis (Durchschnitt von Hoch, Tief und Schluss)

U

CODE	SYNTAX	FUNKTION
Undefined	a = Undefined	Einsetzen einer nicht festgelegten Variable

V

CODE	SYNTAX	FUNKTION
Variation	Variation(price)	Differenz zwischen dem Schluss des letzten Balkens und dem Schluss des aktuellen Balkens in %
Volatility	Volatility[S, L]	Chaikin Volatilität
Volume	Volume[N]	Indikator 'Volumen'
VolumeOscillator	VolumeOscillator[S,L]	Volumen-Oszillator
VolumeROC	VolumeROC[N]	Volumen des 'Rate Of Change'

W

CODE	SYNTAX	FUNKTION
WeightedAverage	WeightedAverage[N](price)	Indikator 'Gewichteter Gleitender Durchschnitt'
WeightedClose	WeightedClose[N]	Durchschnittswert von (2*Schluss), (1*Hoch) und (1*Tief)
WEND	Vgl. WHILE/DO/WEND	End-Anweisung für die WHILE-Schleife
WHILE/DO/WEND	WHILE (condition) DO (action) WEND	WHILE-Schleife
WilderAverage	WilderAverage[N](price)	Indikator 'Gleitender Durchschnitt nach Wilder'
Williams	Williams[N](close)	%R vom Williams-Indikator
WilliamsAccumDistr	WilliamsAccumDistr(price)	Indikator 'Akkumulation/Verteilung nach Williams'

X

CODE	SYNTAX	FUNKTION
XOR	a XOR b	Logischer Operator 'ODER exklusiv'

Y

CODE	SYNTAX	FUNKTION
Year	Year[N]	Gibt das Jahr im Format YYYY an
Yesterday	Yesterday[N]	Zeigt den vorherigen Tag im Format YYYMMDD an

Z

CODE	SYNTAX	FUNKTION
ZigZag	ZigZag[Zr](price)	Indikator 'ZigZag' zur Theorie der Eliott Wellen
ZigZagPoint	ZigZagPoint[Zp](price)	Indikator 'ZigZag' zur Theorie der Eliott Wellen in Zp-Punkten berechnet

Weitere

CODE	FUNKTION	CODE	FUNKTION
+	Addition	<>	Differenz
-	Subtraktion	<	'kleiner als'
*	Multiplikation	>	'größer als'
/	Division	<=	'kleiner oder gleich'
=	Gleichheit	>=	'größer oder gleich'

ProRealTime SOFTWARE