NanoTrader-Express

Sprach Referenz

Dokumenten Version 3.4.1

www.fipertec.com

FIPERTEC
Financial Performance Technology

Inhalt

1 Einführung 4
2 Express – Execute Only 4
3 Struktur eines Express-Programms 5
4 Ein Beispiel-Sentimentor 5
5 Express-Sentimentor zu einer Analyse hinzufügen 8
6 Arbeiten mit dem Express Editor 10
6.1 Starten des Express-Editors 10
6.2 Editor Symbolleiste und Kontext-Menü 11
6.3 Weitere Editor-Funktionen 16
6.3.1 Hervorheben identischer Wörter 16
6.3.2 Funktions-Signaturen anzeigen 17
6.3.3 Einrücken von Blöcken 17
6.4 Tastatur-Befehle 17
6.5 Ausführen eines Express Programms 18
6.6 Speicherung von Express-Programmen 18
7 Express Sprach-Elemente 20
7.1 Typen 20
7.2 Schlüsselworte 20
7.3 Ausdrücke 21
7.3.1 Numerische Ausdrücke 21
7.3.2 Relationale Ausdrücke 21
7.3.3 Logische Ausdrücke 22
7.3.4 String Ausdrücke 22
7.4 Variablen Deklarationen 22
7.5 Input Variablen 25
7.6 Zugriff auf Variablen und Elemente aus Serien 26
7.7 Zugriff auf Array-Elemente 27
7.8 Zuweisungen 27
7.9 Zuweisung von Sentimenten 27
7.10 Vordefinierte Serien 28
7.11 Importieren einer Series aus einem Sentimentor 28
7.12 Importieren eines Arrays aus einem Sentimentor 31
7.13 Importieren der Preisdaten eines anderen Symbols 32
7.14 Datum-und Zeit-Konstanten 32

7.15 Anweisungen 33
7.16 Kontroll-Strukturen 33
7.16.1 if 33
7.16.2 If then else 33
7.16.3 While Schleife 34
7.16.4 For Schleife 34
7.17 Performanz bei der Berechnung von Sentimentoren 35
7.18 Interpretation – Bestimmung der Sentimente 36
7.18.1 Verwendung der standardisierten Interpretations-Schemata 37
7.18.2 Explizite Programmierung der Interpretation 37
7.19 Zeichnen 38
7.20 Mehr zum Thema Farben 39
7.20.1 Duale Farben 39
7.20.2 Syntax der Definition dualer Farben 40
7.20.3 Syntax der Definition dualer Farben 41
7.20.4 Angabe der Deckkraft für Füllfarben, Highlight() und Annotate() 41
8 Ein Beispiel-Blocker 42
9 Ein Beispiel-Stop 43
10 Ein Beispiel-Stop/Taktik mit Intraperioden-Updates 44
11 Debuggen von Express-Programmen 46
11.1 Haltepunkte setzen 46
11.1.1 Im Editor 46
11.1.2 Haltepunkte durch Funktionsaufrufe 47
11.2 Variablen inspizieren während eines Programm-Halts 49
11.3 Hervorhebung von geänderten Variablen 50
11.4 Fortsetzen der Code-Ausführung von einem Haltepunkt aus 50
11.5 Einige Tipps & Tricks zum Debuggen 51
11.6 Verwenden von DDE zur Simulation bestimmter Datenkonstellationen 52
12 Express-Sentimentoren verschlüsseln 52
13 Verfügbare Funktionen und Prozeduren 53

1 Einführung

Herzlich Willkommen zu NanoTrader-Express!

NanoTrader-Express ermöglicht die Erstellung von Sentimentoren, preisbasierten Stops und Blockern, die auf die gleiche Art und Weise verwendet werden können, wie die direkt in NanoTrader integrierten Sentimentoren. So können Express-Sentimentoren mit beliebigen anderen Sentimentoren zu Analysen kombiniert und natürlich optimiert werden. Die Kombination von NanoTrader mit NanoTrader-Express bildet somit eine einzigartige Umgebung zum Spezifizieren, Optimieren, Testen und Umsetzen von Handelsideen.

NanoTrader-Express kann auch ohne das Modul „Handelssysteme“ äußerst nützlich eingesetzt werden. In diesem Fall kann Express verwendet werden, um Indikatoren zu zeichnen, um den Chart automatisiert mit Symbolen oder Texten zu versehen oder um sich durch Alarme und Meldungen auf besondere Konstellationen aufmerksam machen zu lassen.

Dieses Dokument beschreibt die Sprache NanoTrader-Express. Es wird eine grundlegende Kenntnis in der Entwicklung kleinerer Programme oder Skripte vorausgesetzt. Leser, die noch keinerlei Erfahrung mir den Konzepten der Programmierung haben, sollten zunächst die ersten Seiten eines einführenden Buchs zu diesem Thema lesen, z.B. über Visual Basic, Pascal, Excel Programmierung oder über die Easy Language der TradeStation. Sobald die wesentlichen Begriffe wie *Variablen*, *Schleifen* oder *bedingte Ausdrücke* verstanden sind, wird die Arbeit mit NanoTrader-Express sehr einfach sein.

Der Leser sollte weiterhin mit der Terminologie vertraut sein, die in der Anleitung „NanoTrader – Charting & Trading“ verwendet wird.

Anwender, die Erfahrung mit der Entwicklung von Indikatoren oder Handelssystemen haben, sollten sich noch einmal den “Sentimentor”-basierten Ansatz von NanoTrader vergegenwärtigen, der auch in NanoTrader-Express Anwendung findet. Express beinhaltet keine Anweisungen wie „buy at open“, vielmehr stellt ein in Express entwickelter Sentimentor einen Baustein dar, der Stimmungen generiert. Die Stimmungen dieses Bausteins werden in einer Analyse über den MetaSentimentor mit anderen Stimmungen kombiniert und schließlich in entsprechende Handelssignale umgesetzt.

Für besonders komplexe oder rechenintensive Sentimentoren steht ein separates API zur Verfügung, das die Einbindung von Sentimentoren als DLLs ermöglicht, die in Sprachen wie C oder C++ erstellt wurden.

2 Express – Execute Only

Um aktiv mit NanoTrader in Express zu programmieren, wird eine entsprechende Lizenz benötigt. Allerdings kann ein Express-Sentimentor von jedem NanoTrader-Anwender *ausgeführt* und auch *angesehen* werden, ohne dass dafür eine spezielle Lizenz nötig wäre.

Damit ein Express-Sentimentor von „Execute Only“-Anwendern ausgeführt werden kann, ist es nötig, dass der Autor des Sentimentors diesen einmal mittels des Express-Editors speichert. Dabei wird dem Sentimentor automatisch ein Wasserzeichen hinzugefügt, das für die Ausführung durch „Execute Only“-Anwender benötigt wird.

Wir ermutigen alle NanoTrader-Express-Entwickler, ihre Sentimentoren anderen Usern zur Verfügung zu stellen.

3 Struktur eines Express-Programms

Ein Express-Programm zur Berechnung eines Sentimentors hat stets folgende Struktur:

- Deklaration und Initialisierung aller für die Berechnung verwendeten Variablen (`Variable Declarations Section`)
- pro Periode Berechnung der für die Stimmungsgebung benötigten Daten (`Calculation Section`)
- pro Periode Berechnung der ausgewiesenen Stimmung, d.h. wie sind die berechneten Daten hinsichtlich ihres Einflusses auf die Signalgebung zu *interpretieren* (`Interpretation Section`)
- Definition der darzustellenden Charts und Linien (`Plot Section`)

4 Ein Beispiel-Sentimentor

Die Struktur eines typischen Express-Sentimentors soll am Beispiel eines einfachen Sentimentors erläutert werden. Der vorgestellte Sentimentor berechnet zunächst einen exponentiell geglätteten gleitenden Durchschnitt der Schlusskurse (EMA) des geladenen MasterCharts. Die Sentimente werden dann auf Basis von Kreuzungen des EMA mit dem Schlusskurs festgelegt. Dieser Sentimentor ist Teil der NanoTrader-Express Distribution.

(Hinweis: Der vorgestellte Sentimentor hat im wesentlichen didaktischen Charakter. In Analysen würde man den vordefinierten Sentimentor “Crossing MA” bzw. die Funktion MovingAverage() verwenden.)

```
//(c) Fipertec                                    1.

                                                  2.
Express EMA                                       3.
Vars                                              4.
input $span (1, 200, 10);                         5.
numeric factor (0);                               6.
series ema;

                                                  7.
Calculation                                       8.
factor = 2 / ($span + 1);
```

```
if close[1] = void then                                    9.
  ema = close;                                             10.
else
  ema = factor * close + (1 - factor) *                    11.
ema[1];
                                                           12.
interpretation TriggerLine(close, ema);
                                                           13.
plot (ema, blue, 2);                                       14.
plot (close, black, 1);
```

Erläuterung:

1. Die Zeichen nach einem Doppel-Slash `//` bis zum Ende der Zeile werden als Kommentar aufgefasst. Um Kommentare an beliebigen Stellen im Code einzuführen, müssen geschweifte Klammern verwendet werden:
   `{ das ist ein Kommentar }`. In geschweifte Klammern eingeschlossene Kommentare können auch mehrere Zeilen umfassen. Dies ist insbesondere nützlich, um zu Testzwecken eine Reihe von Anweisungen zu deaktivieren.
2. Ein Programm startet immer mit `Express <name>`, wobei `name` der Name des Express-Sentimentors ist, der im Dynamite-Dialog angezeigt wird.
3. Durch das Schlüsselwort `Vars` wird die `Variable Declarations Section` eingeleitet.
4. Diese Zeile deklariert eine ganzzahlige Variable. Im folgenden werden statt ganzzahlig auch die Bezeichnungen *integer* oder *int* verwendet.
   Die definierte Variable `$span` kann in der Optimierung von NanoTrader verändert werden. Die untere Grenze ist dabei 1, die obere Grenze 200 und der initiale Wert ist 10.
   Der Name der Variablen, span, wird automatisch im Dynamite-Dialog angezeigt und kann dort auch wie gewohnt manuell geändert werden.
   Das `$`-Zeichen wird verwendet, um überall im Code deutlich zu machen, dass es sich bei der Variablen um eine Optimierungs-Variable handelt.
5. `factor` ist als *numerische* Variable definiert. Eine numerische Variable kann sowohl ganze Zahlen (integers) aufnehmen als auch Gleitkommazahlen, wie 3.75. Eine Gleitkommazahl wird im folgenden auch als *float* bezeichnet.
   Durch die Notation `(0)` wird die Variable mit 0 initialisiert. Da Express numerische Variablen automatisch mit 0 initialisiert, könnte auf die Angabe von `(0)` auch verzichtet werden.
   Express unterscheidet nicht zwischen Groß- und Kleinschreibung, d.h., `factor`, `Factor`, `FActOR` referenzieren alle dieselbe Variable. Auch

bei Schlüsselworten kann beliebig Gross-und Kleinschreibung verwendet werden.

6. `ema` wird als eine `series` von numerischen Werten definiert. Eine `series` hat automatisch die gleiche Anzahl Elemente wie die Preisdaten des analysierten MasterCharts. Die Elemente einer `series` werden automatisch mit 0 initialisiert. Auch hier kann der `(wert)`-Mechanismus verwendet werden, um alle Elemente der `series` mit `wert` zu initialisieren.

7. Das Schlüsselwort `Calculation` leitet den Abschnitt ein, der die Anweisungen zur Berechnung der für den Sentimentor benötigten Daten enthält. Diese Anweisungen werden für jede Periode der geladenen Preisdaten ausgeführt, wobei mit der ältesten – am weitesten „links“ stehenden – Periode begonnen wird.

8. Der `factor` für die Berechnung des EMA wird basierend auf der Input-Variablen `$span` ermittelt. Für derartige *mathematische Operationen* stehen die üblichen Operatoren `+, -, *, /` zur Verfügung. Zur Gruppierung von Ausdrücken können Klammern `()` verwendet werden.

9. Express stellt eine Reihe von `series`-Variablen automatisch zur Verfügung, u.a., `close, open, high, low, volume,` die lesenden Zugriff auf die Daten des zu analysierenden MasterCharts ermöglichen.
   Um den EMA-Wert für die aktuelle Periode zu ermitteln, muss auf den EMA-Wert der Vorgängerperiode zugegriffen werden. Zur Referenzierung zurückliegender Daten verwendet Express einen Inidizierungsmechanhismus: `close[1]` bezeichnet den Schlusskurs der vorigen Periode. Allgemein bezeichnet `close[n]` den Schlusskurs vor `n` Perioden und `close` ist lediglich eine Kurzschreibweise für `close[0]`.

   Angenommen der EMA-Wert soll für die erste Periode bestimmt werden. In diesem Fall gibt es keine Vorgängerperiode und `close[1]` ist somit nicht verfügbar. Bei Zugriff auf nicht verfügbare Daten liefert Express die Konstante `void`, was soviel wie „nicht definiert“ oder „nichts“ bedeutet.
   Die *bedingte Anweisung* `if` führt die nach `then` aufgeführte Anweisung aus, falls die `if`-Bedingung erfüllt ist. Ansonsten wird die unter `else` angegebene Anweisung ausgeführt. Falls mehrere Anweisungen auszuführen wären, müssten diese mit den Schlüsselworten `begin` und `end` zusammengefasst werden.

10. Dem ersten Element der `ema` Serie wird ein Wert zugewiesen.

11. Der `ema` für die aktuelle Periode wird berechnet.

12. Express stellt eine Reihe vordefinierter *Interpretations-Schemata* zur Verfügung, die Sentimente berechnen und frei konfiguriert werden können. Das Schema *TriggerLine* erfüllt genau den von uns gewünschten Zweck und ermittelt die Sentimente basierend auf den Schnittpunkten zweier Linien.
    Falls keines der zur Verfügung stehenden Interpretations-Schemata verwendet werden kann, können die Sentimente auch explizit in der

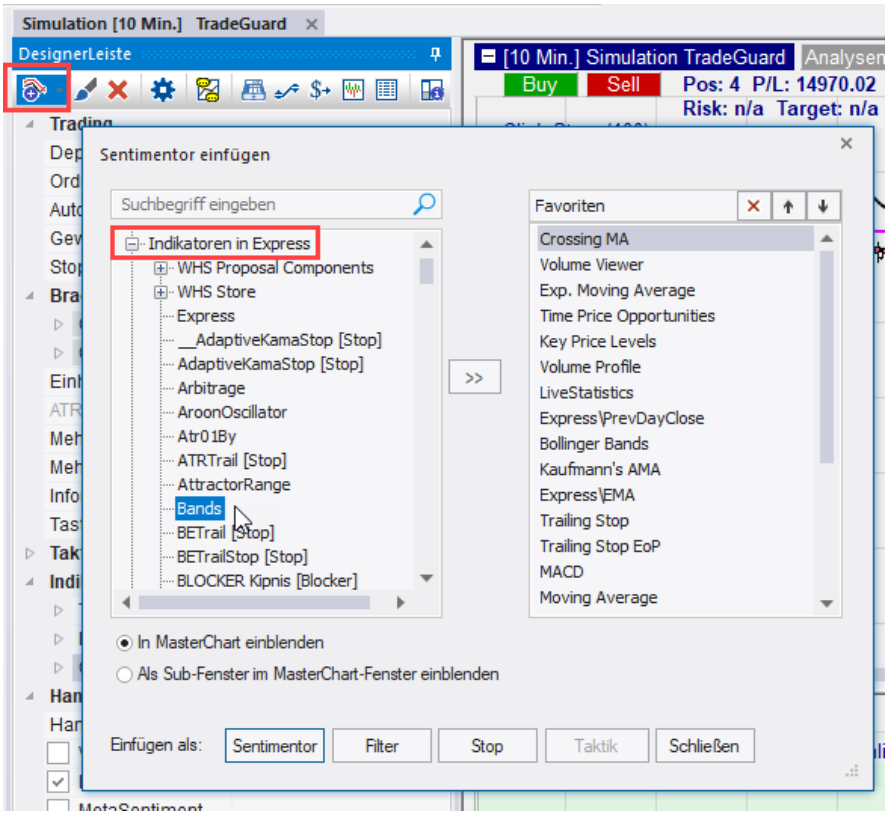
`interpreation section` programmiert werden. Die Sentimente müssen dann der vordefinierten Serie `sentiment` zugewiesen werden. Die Anweisungen zur Berechnung der Sentimente müssen in diesem Fall mit `begin` und `end` zusammengefasst werden:

```
interpretation
begin
  if … then
    sentiment = 100;
  else
    …
end
```

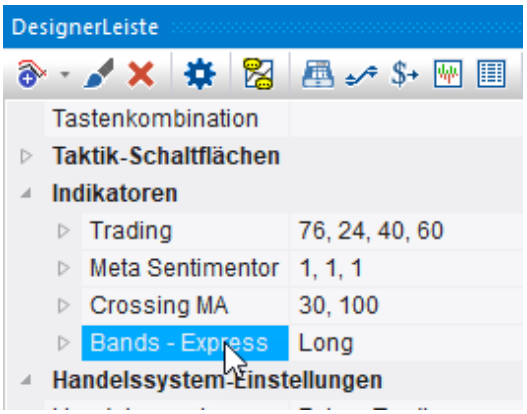
13. Die berechnete `ema` Serie wird in blau mit einer Strichstärke von 2 gezeichnet.
14. Die Schlusskurse werden in schwarz in einer Strichstärke von 1 gezeichnet.

5 Express-Sentimentor zu einer Analyse hinzufügen

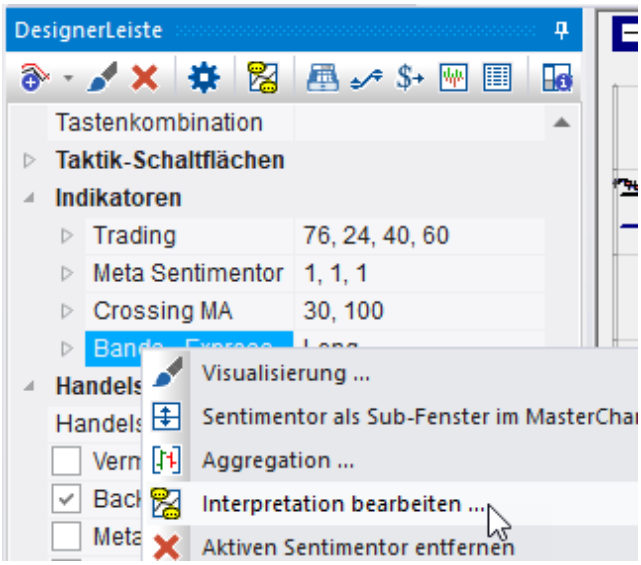
Ein Express-Sentimentor kann über den Sentimentor einfügen-Dialog einer Analyse hinzugefügt werden:



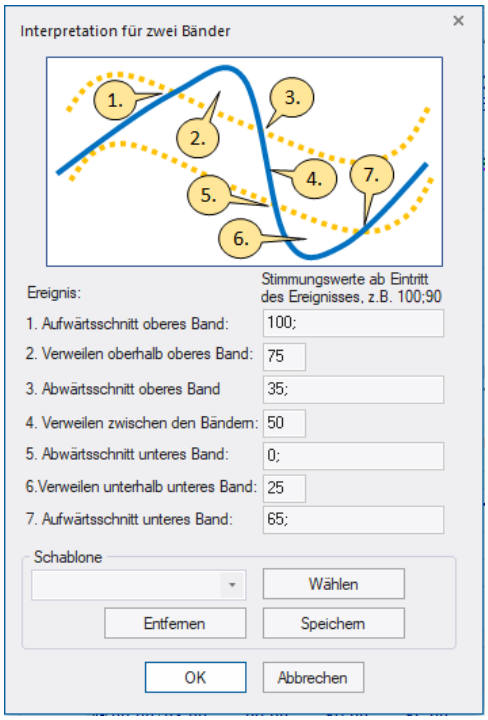
Um den Code eines Express-Sentimentors im Express-Editor zu bearbeiten, doppelklicken Sie auf die entsprechende Zeile in der DesignerLeiste:



Falls der Express-Sentimentor ein vorgefertigtes Interpretations-Schema verwendet, kann dieses konfiguriert werden, indem das Icon geklickt wird oder indem über das Kontextmenü des Express-Sentimentors der Menüpunkt Interpretation bearbeiten gewählt wird:



Dadurch wird der entsprechende Sentiment-Editor geöffnet:

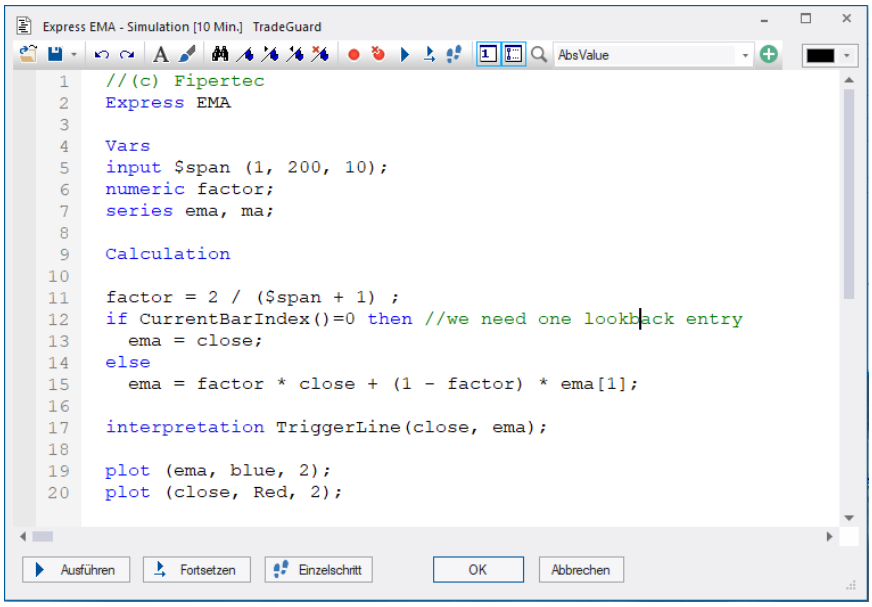


Eine genaue Beschreibung der Arbeit mit den vorgefertigten Interpretations-Schemata findet sich in der NanoTrader *Bedienungsanleitung*.

6 Arbeiten mit dem Express Editor

6.1 Starten des Express-Editors

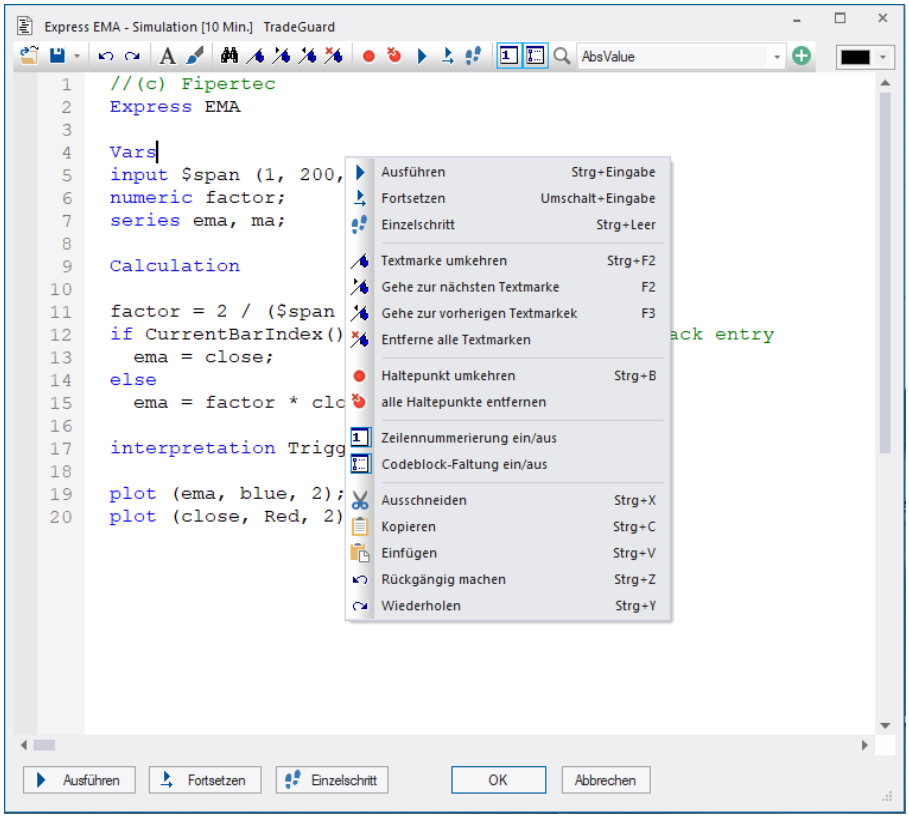
Der Express-Editor wird durch Doppelklick auf eine Express-Sentimentor in der DesignerLeiste gestartet:



Der Editor stellt die Schlüsselworte in blau und Kommentare in grün dar. Alle Farben, die vom Editor verwendet werden, können im Farb-Manager eingestellt werden.

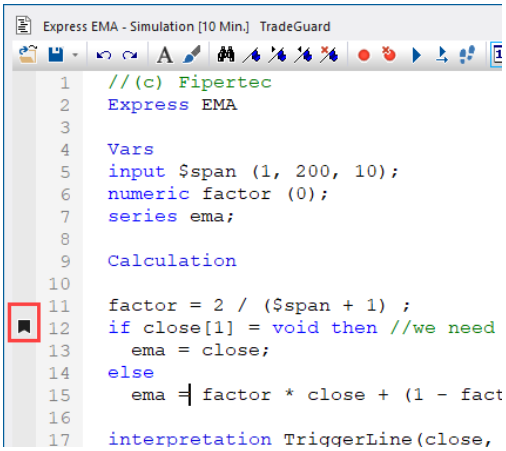
6.2 Editor Symbolleiste und Kontext-Menü

Auf die meisten Funktionen des Editors kann über die Symbolleiste oder das Kontextmenü zugegriffen werden. Außerdem gibt es eine Reihe von Tastenkombinationen, die eine schnelle Ausführung verschiedener Aktionen ermöglichen. Die jeweiligen Tastenkombinationen werden auf der rechten Seite des Kontextmenüs angezeigt:

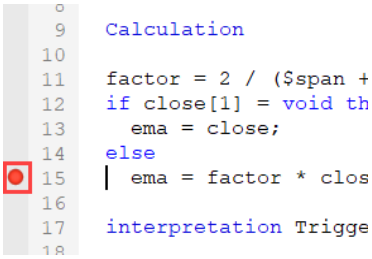


- **Open**
  Ersetzt den aktuellen Code durch den der ausgewählten Datei. Siehe
- **Speichern/Speichern unter**
  Speichert den Code in einer externen Datei. Siehe Abschnitt "Speicherung von Express-Programmen" für Details.
- **Rückgängig/Wiederherstellen**
  Die letzte(n) Textoperation(en) rückgängig machen oder Wiederherstellen.
- **Zeichensatz wählen**
  Den Zeichensatz für die Darstelleung des Codes wählen.

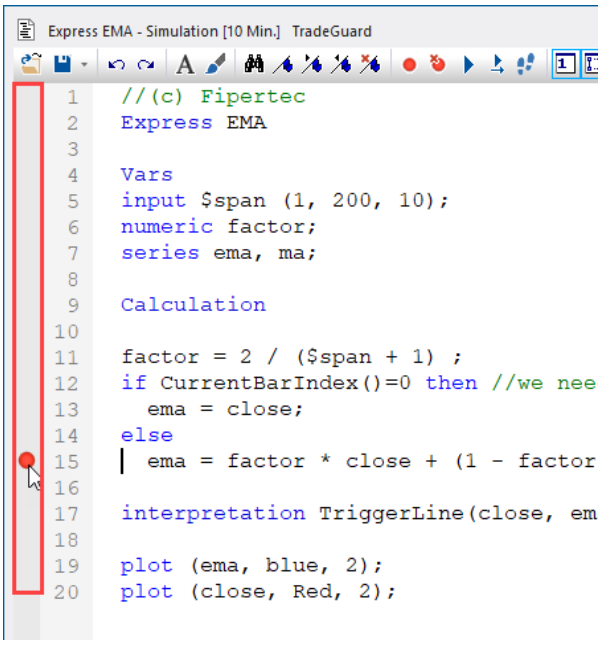
- **Farben einstellen**
  Farben für die Anzeige, das Hervorheben und Markieren des Textes einstellen.
- **Suchen & Ersetzen**
  Öffnet einen Dialog zum Suchen oder Ersetzen von Text.
- **Textmarke**
  Ermöglicht es, Textmarken an wichtigen Stellen zu setzen und schnell zu ihnen zu navigieren.



- **Haltepunkte**
  Haltepunkt setzen oder entfernen. (Weitere Details unter "Debuggen von Express-Programmen".)

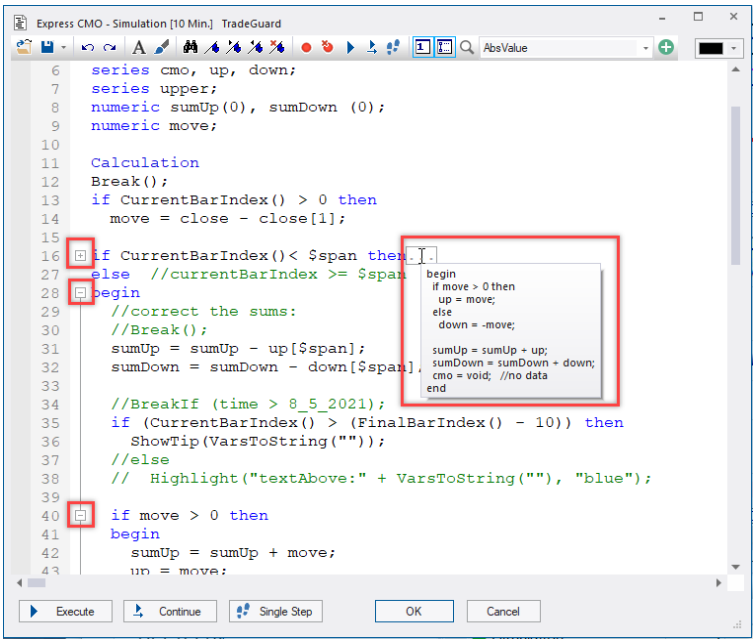


Ein Haltepunkt kann auch einfach durch Klick in den Markierungsbereich gesetzt werden:



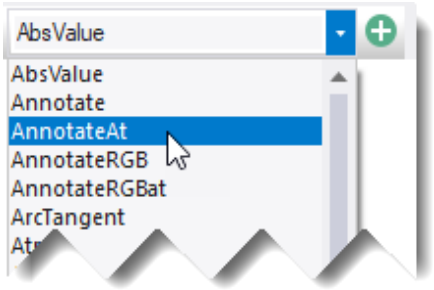
Um einen einzelnen Haltepunkt zu entfernen, klicken Sie ihn im Markierungsbereich an oder platzieren Sie den Cursor in der Zeile daneben und verwenden Sie die Schaltfläche in der Symbolleiste, das Kontextmenü oder Strg-B.
Um alle Haltepunkte zu entfernen, klicken Sie in der Symbolleiste oder im Kontextmenü.

- **Zeilennummern**
  Ein/Ausschalten der Zeilennummerierung.
- **Codeblock-Faltung**
  Ermöglicht das Ein- und Ausblenden von Codeblöcken. Dies kann hilfreich sein, wenn Sie an einem sehr großen Programm arbeiten. Codeblöcke, die ein- oder ausgeblendet werden können, werden durch die Schaltflächen bzw. dargestellt.
  Ein eingeklappter Block wird durch dargestellt. Wenn Sie mit der Maus auf die Punkte zeigen, wird ein Popup mit den eingeklappten Codezeilen angezeigt.



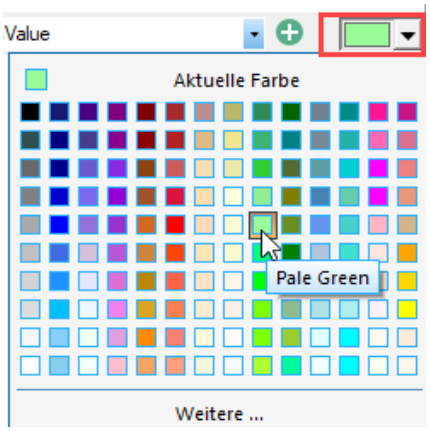
- **Code Analyse**
  Prüft den Code auf Effizienz sowie auf unbenutzte Serien und Variablen. Siehe auch Abschnitt "Performanz bei der Berechnung von Sentimentoren" für einige Hinweise zur Erstellung effizienten Codes.

- **Funktionen-Helfer**

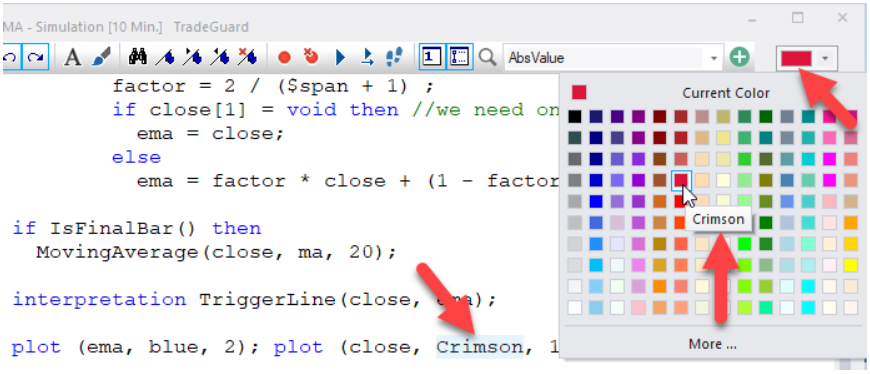


  Die Dropdown-Liste zeigt eine Liste der verfügbaren Express-Funktionen an. Um die Funktion in den Text einzufügen, wählen Sie sie zuerst aus und klicken Sie dann die Schaltfläche . Um die Signatur der Funktion anzuzeigen, fahren Sie mit der Maus über die Funktion im Code.

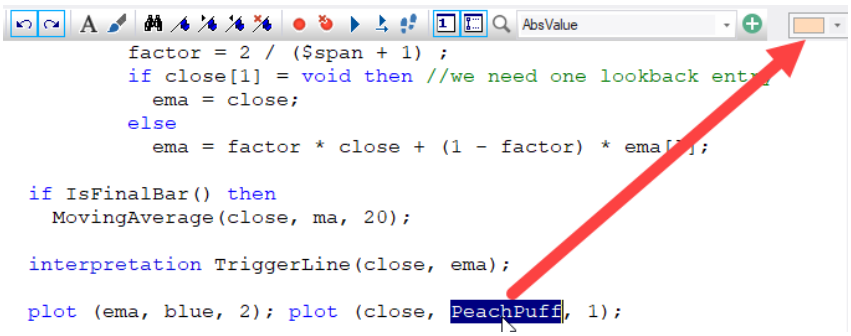
- **Farbwähler & Anzeiger**



Das Festlegen ansprechender Farben für geplottete Elemente ist wichtig, um das Beste aus Ihrem Express-Code herauszuholen.
Um eine Farbe *auszuwählen*, öffnen Sie das Dropdown-Menü und wählen eine der vordefinierten Farben aus. Sobald eine Farbe ausgewählt wurde, wird ihr Name an der aktuellen Cursorposition eingefügt:

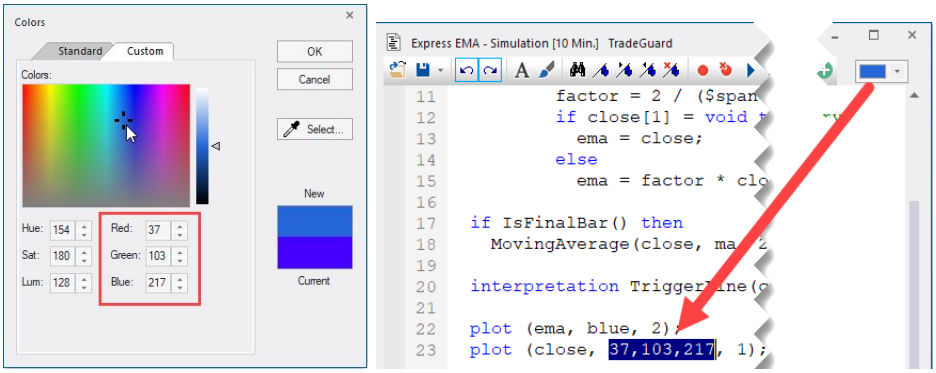


Um die Farbdarstellung eines Farbnamens zu sehen, doppelklicken Sie auf ihn im Code. Die Farbe wird dann angezeigt:



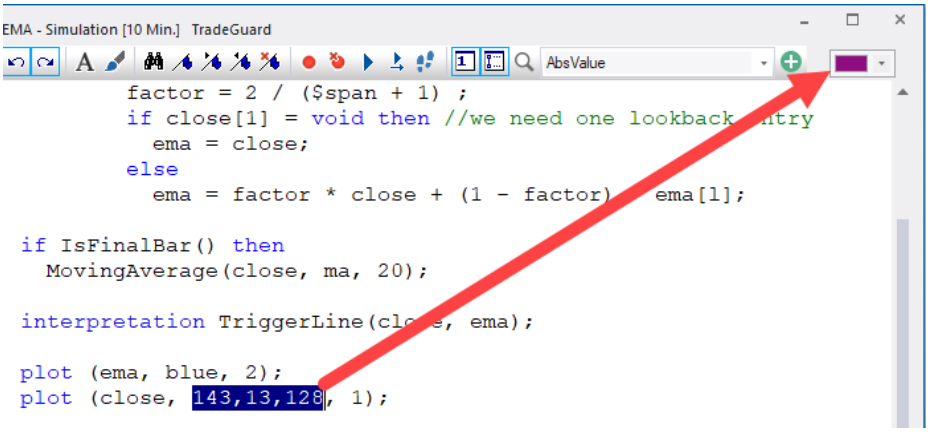
Express unterstützt auch die RGB-Notation zur Definition von Farben, so dass jede potenzielle Farbe definiert werden kann. Wählen Sie im Dropdown-Menü des Farb-Wählers die Option Mehr...

Wählen Sie dann eine beliebige Farbe:



Die RGB-Notation wird automatisch im Text eingefügt.

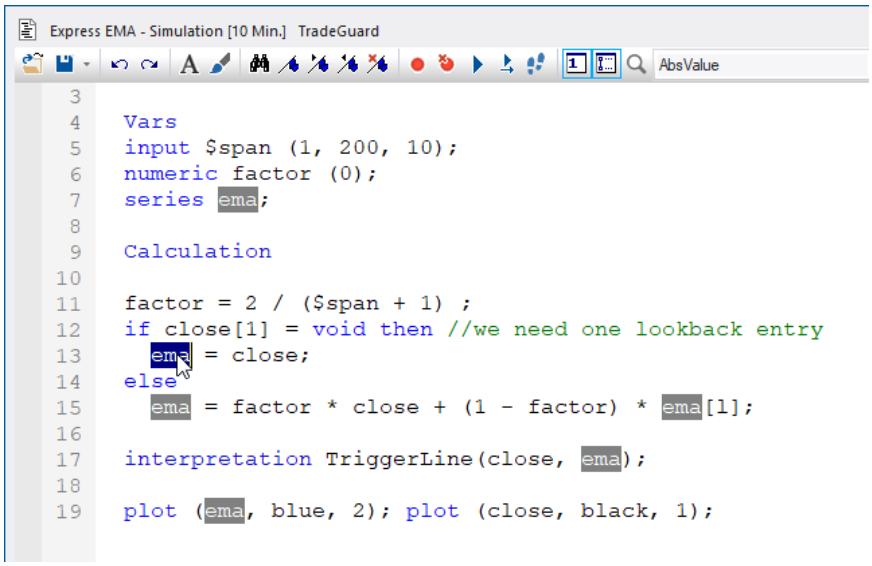
Umgekehrt können Sie eine RGB-Notation im Text markieren und die zugehörige Farbe wird dann angezeigt:



6.3 Weitere Editor-Funktionen

6.3.1 Hervorheben identischer Wörter

Wenn Sie ein Wort doppelklicken, werden alle identischen Wörter im Text markiert. Dies ist sehr hilfreich, um schnell zu erkennen, wo eine Variable oder Funktion verwendet wird:

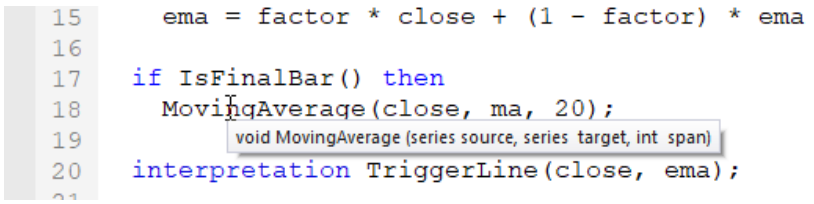


Im obigen Beispiel wurde die Variable `ema` doppelt angeklickt, wodurch alle Vorkommen von `ema` markiert wurden.

Um die Markierungen zu entfernen, drücken Sie die Escape-Taste oder klicken Sie auf eine leere Stelle im Editor.

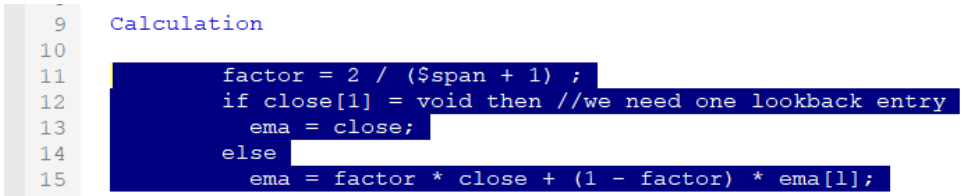
6.3.2 Funktions-Signaturen anzeigen

Bewegen Sie die Maus über eine Funktion, um deren Signatur zu sehen:



6.3.3 Einrücken von Blöcken

Um einen Zeilenblock einzurücken, markieren Sie ihn mit der Maus und drücken dann die Tabulatortaste. Drücken Sie Shift-Tab, um die Einrückung aufzuheben:



6.4 Tastatur-Befehle

Wo möglich verwendet der Editor die unter Windows üblichen Tastaturbefhehle.

Strg-C	kopiert den ausgewählten Text in die Zwischenablage
Strg -V	fügt Text aus der Zwischenablage ein
Strg -A	wählt den gesamten Text aus

Strg-F2	Textmarke für aktuelle Zeile umkehren
F2	gehe zu nächster Textmarke
F3	gehe zu vorheriger Textmarke
Strg -B	Haltepunkt für aktuelle Zeile umkehren
Strg -S	speichert den Text in einer Datei
Strg -Z	macht die letzte Textaktion rückgängig
Strg -Y	Wiederherstellen nach Rückgängig machen
Strg -<Return>	führt das Programm von Beginn an aus
Strg -<Space>	Einzelschritt, d.h. Halte in nächster Zeile
Shift<Return>	setzt die Ausführung vom aktuellen Haltepunkt aus fort

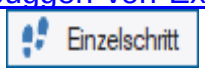
Hinweis: Um versehentliche Aktionen zu vermeiden, die durch Tastenkombinationen ausgelöst werden könnten, wie z. B. das Platzieren oder Löschen von Aufträgen, ist *die **Hotkey-Behandlung** von NanoTrader deaktiviert, während ein Express-Editor das aktive Fenster von NanoTrader ist.*

6.5 Ausführen eines Express Programms

Wenn Sie auf die

-Schaltfläche klicken, wird das Express-Programm ausgeführt und die Ergebnisse in Form von Plots und Signalen im Chart angezeigt. Im Falle von Programmfehlern werden entsprechende Fehlermeldungen erzeugt.

Im Abschnitt "Debuggen von Express-Programmen" wird auf die Schaltflächen

und 

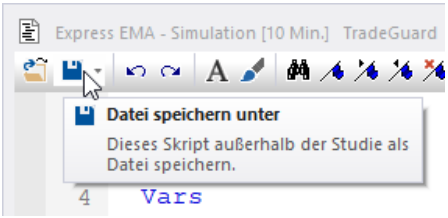
eingegangen.

6.6 Speicherung von Express-Programmen

Ein im Editor geöffnetes Express-Programm ist immer mit einer Studie verknüpft.

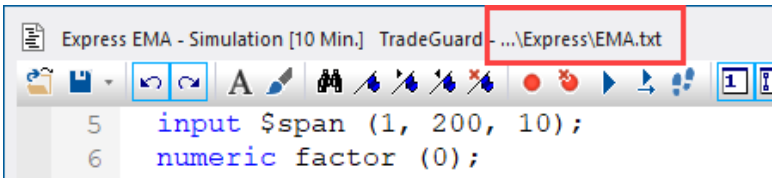
Beim Verlassen des Express-Editors durch Anklicken der OK-Taste wird das aktuelle Programm an den Sentimentor übergeben, aus dem es stammt. Wenn Sie Ihre Änderungen sichern möchten, *müssen Sie die Studie speichern!*

Sehr oft soll der Code aber auch als Sentimentor in anderen Studien verwendet werden. Um das zu erreichen, können Sie den Code als separate Datei speichern, indem Sie auf das Speichern-Symbol klicken:



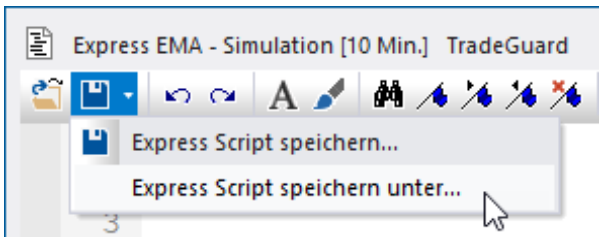
Wenn Sie das erste Mal auf das Speichersymbol klicken, werden Sie nach einem Dateinamen gefragt. Wenn Sie anschließend erneut auf das Speichersymbol klicken oder Strg-S drücken, wird diese Datei überschrieben.

Der aktuell aktive Dateiname wird in der Titelleiste des Editor-Fensters angezeigt:



Um Platz zu sparen, wird der Pfad des Installationsverzeichnisses weggelassen und durch drei Punkte repräsentiert.

Wenn Sie den Dateinamen ändern möchten, klicken Sie auf den Dropdown-Pfeil neben dem Speichersymbol und wählen Sie "Express Script speichern unter...":



Wenn Sie die Datei im Express-Verzeichnis der NanoTrader-Installation speichern (das ist der automatisch vorgeschlagene Speicherort), wird das Programm als Sentimentor im Sentimentor hinzufügen-Dialog angezeigt.

Es ist guter Stil, den Namen des Express-Programms und den Namen der Textdatei identisch zu wählen. Angenommen das Express-Programm beginnt mit

```
Express UltimateSenti
```

dann wird `UltimateSenti` der Name des Sentimentors. Dieser Name erscheint in der DesignerLeiste und in der Legende des Chart-Fensters. Wenn das Programm als Datei gespeichert wird, sollte daher auch der Name “UltimateSenti.txt” verwendet werden, auch wenn dies nicht vom Editor erzwungen wird.

7 Express Sprach-Elemente

7.1 Typen

Express ist eine sogenannte *typisierte* Sprache, d.h. jedes Element, das einen Wert repräsentiert, hat einen zugeordneten Typ. Diese Typisierung erlaubt es, eine große Anzahl potenzieller Programmierfehler bereits vor Ausführung des Programms zu ermitteln.

Die folgenden Typen stehen in Express zur Verfügung:

- `numeric`
  Ein numerischer Wert kann ein float sein (z.B. 3.75) oder ein integer (z.B. 10). Falls erforderlich konvertiert Express automatisch float-Werte in integer-Werte, in dem die Nachkommastellen abgeschnitten werden.
- `series`
  Eine `series` ist eine Serie von Werten vom Typ `numeric`. Angenommen der analysierte MasterChart umfasst 200 Perioden, dann umfasst jede Serie automatisch ebenfalls 200 Werte. Falls der MasterChart mit einer Realtime-Datenquelle verbunden ist, werden automatisch alle Serien vergrößert, sobald eine neue Periode durch eingehende Ticks eröffnet wird.
- `Array`
  Ein `Array` ist eine feste Anzahl Elemente vom Typ `numeric`. Im Gegensatz zum Datentyp `series` ändert sich die Größe eines Arrays nicht, falls neue Perioden im MasterChart erscheinen.
- `string`
  Ein `string` ist eine Kette von Zeichen, z.B. `Hello World!`. Wenn ein `string` als Konstante verwendet wird, müssen die Zeichen in Anführungsstriche eingefasst werden: `"Hello World!"`
- `bool`
  Ein `bool`-Wert oder *boolescher* Wert umfasst den Wertebereich `true` und `false`.
- `time`
  Der Typ `time` wird verwendet, um ein Datum oder eine Zeit zu repräsentieren:
  ```
  if timeOpen < 10:00 then sentiment = 50;
  if date = 2_8_2002 then sentiment = 100;
  ```

7.2 Schlüsselworte

Express verwendet eine Reihe sogenannter *Schlüsselworte*, die in einem Express-Programm jeweils eine wichtige Rolle spielen. Ein Schlüsselwort kann nicht als Name einer Variablen verwendet werden.

Die derzeitig verwendeten Schlüsselwort sind:

```
express, sentimentor, blocker, stop, vars,
calculation,interpretation, numeric, bool, export, info,
senti_block, senti_flat, senti_pass, series, string, if,
then, else, begin, end, for, to, downto, while, void, and,
or, plot, plotband, plotcrossinglines, plotline,
plotcandles, plotbars.
```

Die folgenden Schlüsselworte sind für eine zukünftige Verwendung vorgesehen:

```
function, procedure, import, plothistogram
```

7.3 Ausdrücke

Ein *Ausdruck* ist eine Kombination eines *Operators* mit *Operanden*, z.B.

```
5 + 3
```

wobei `5` und `3` die Operanden sind und `+` der Operator.

Auch der Wert einer Variablen, einer Serie und der Rückgabewert einer Funktion sind Ausdrücke.

In Express werden drei Typen von Ausdrücken unterschieden:

- numerische Ausdrücke
- boolesche Ausdrücke (logische Verknüpfungen)
- string Ausdrücke.

die im folgenden näher erläutert werden.

7.3.1 Numerische Ausdrücke

Numerische Werte können mit den Operatoren `+`, `-`, `*`, `/` in ihrer üblichen mathematischen Bedeutung verknüpft werden, z.B.

```
5 * close - 3*close[1];
```

Um Teilausdrücke zu gruppieren können Klammern verwendet werden:

```
(high + low + close) / 3
```

7.3.2 Relationale Ausdrücke

Das Ergebnis eines relationalen Ausdrucks ist einer der Wahrheitswert `true` oder `false`, z.B.

```
close > open.
```

Die verfügbaren relationalen Operatoren sind:

Operator	***Bedeutung***
>	grösser als
<	kleiner als
=	gleich (identisch mit)
>=	grösser oder gleich als
<=	kleiner oder gleich als

<>	ungleich (nicht identisch mit)

Relationale Ausdrücke können mit numerischen, time und string-Werten formuliert werden.

Bei string-Werten werden die Operatoren bzgl. der sogenannten *lexikographischen Ordnung* interpretiert („Telefonbuch-Sortierung“), d.h. `"abc"` ist kleiner als `"xyz"` und `"abd"` ist größer als `"abc"`.

7.3.3 Logische Ausdrücke

Ein *logischer Ausdruck* ist eine Kombination von Ausdrücken, die als Ergebnis einen Wahrheitswert `true` oder `false` liefern, z.B.:

```
(close > open) and (volume <= 1000)
```

```
(close > close[1]) and ((open > close) or (volume > 1000))
```

Hinweis: Es sollten immer Klammern verwendet werden, um die exakte Gruppierung der Teilausdrücke hervorzuheben. Dies hilft nicht nur der Lesbarkeit des Codes sondern auch bei der Vermeidung von Programmierfehlern.

7.3.4 String Ausdrücke

Der einzige Operator für die Verknüpfung von string-Werten ist `+`. Dieser liefert als Ergebnis die Konkatenation seiner Operanden, z.B.

```
"Hello "+ "World!"
```

ergibt den neuen String `"Hello World!"`.

7.4 Variablen Deklarationen

Alle in einem Express-Programm verwendeten Variablen müssen in `Variable Declarations Section` deklariert werden, indem ihr Typ und ihr Name angegeben werden. Der Name muss mit einem Zeichen aus dem Alphabet beginnen, z.B. `numeric gewicht;`

Der initiale Wert einer Variablen kann in der Deklaration mit angegeben werden:

```
numeric gewicht (0.5);
```

Um mehrere Variablen des gleichen Typs zu deklarieren kann die folgende Schreibweise verwendet werden:

```
numeric gewicht, faktor, delta;
```

Hinweis: Verwenden Sie stets Namen, die die Bedeutung der Variablen hervorheben. Dies erleichtert die Lesbarkeit des Programms ganz erheblich.

Gültige Typen für Variablen sind `numeric`, `bool`, `series`, und `string`. Falls in der Deklaration kein Initialwert angegeben wird, initialisiert NanoTrader die Variablen wie folgt:

Typ	**Initialwert**
`numeric`	0

`bool`	`false`
`series`	alle Elemente werden auf 0 gesetzt
`array`	alle Elemente werden auf 0 gesetzt
`string`	`""`, d.h. eine leere Zeichenkette

Eine `series`-Variable kann an eine `series` aus einem anderen Express-Sentimentor oder einem Standard-Sentimentor gebunden werden, der Bestandteil der Studie ist. Weitere Details hierzu befinden sich im Abschnitt Importieren einer Series aus einem Sentimentor.

Variablen Exportieren

`numeric` und `string` Variablen können eine `export`-Klausel enthalten:

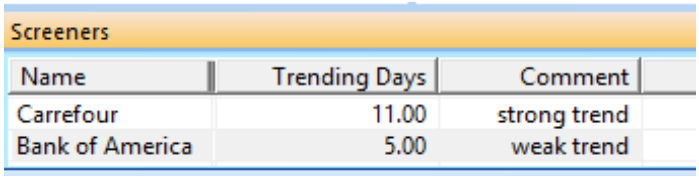
```
export "label;format";
```

Beispiel:

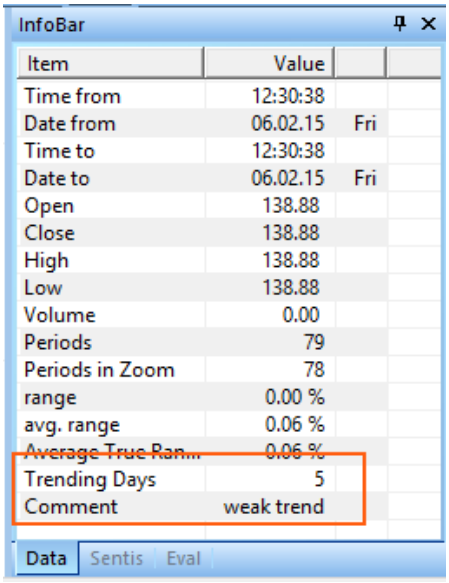
```
numeric result export "Trending Days;%.02f";
string comment export "Comment";
```

Die “;format”-Angabe ist optional. (Siehe Funktion NumericToString() für eine Beschreibung gültiger Format-Anweisungen.)

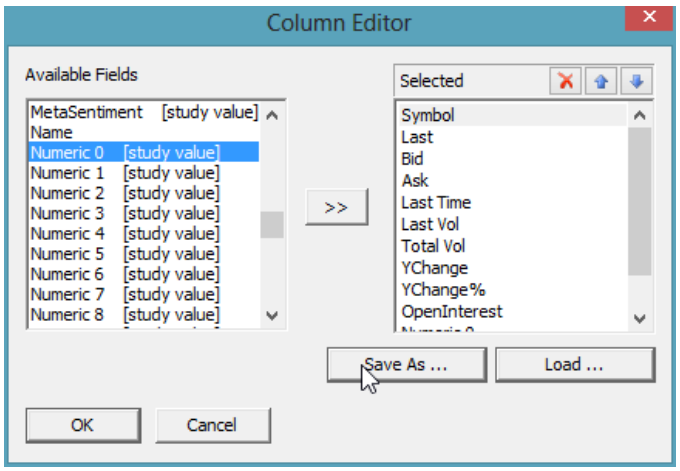
Exportierte Variablen werden in eigenen Spalten von Screener-Ergebnistabellen angezeigt:

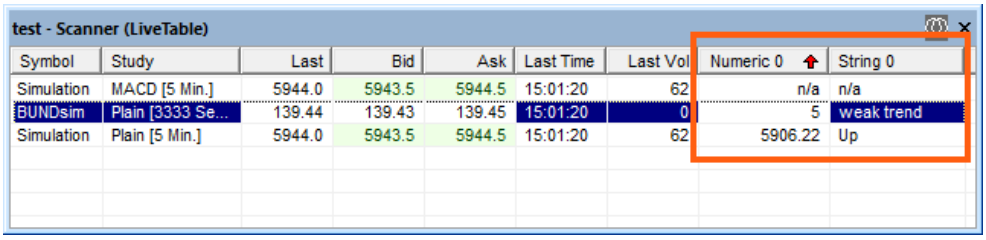


Exportierte Variablen werden auch in der InfoBar einer Studie angezeigt.



Der Konfigurations-Dialog für Scanner (LiveTabellen) ermöglicht die Anzeige exportierter Variablen.
Da eine LiveTabelle unterschiedliche Studien aufnehmen kann, müssen die Spaltenüberschriften allgemein gehalten werden.
Bis zu zehn exportierte Numeric/Series-Variablen und zehn exportierte Strings können angezeigt werden.





7.5 Input Variablen

Die sogenannten `input` Variablen sind numerische Variablen, die als Parameter im Dynamite-Dialog erscheinen und dort sowohl manuell als auch durch die Optimierung variiert werden können. Aus diesem Grund benötigt ihre Definition neben einem Initialwert auch den kleinsten und größten zulässigen Wert. Zur Deklaration einer input Variablen wird folgende Syntax verwendet:

`input $<var Name> (<min Wert>, <max Wert>, <initialer Wert>);`

Beispiel:

```
input $spanne (1, 200, 20);
```

Das `$`-Zeichen wird als Präfix verwendet, um überall im Code unmittelbar sichtbar zu machen, dass es sich bei der Variablen um eine `input` Variable handelt, die von außen modifiziert werden kann und daher von großer Bedeutung für den Sentimentor ist.

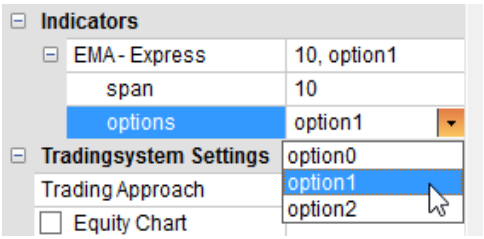
Eine `numeric`-Variable kann so definiert werden, dass sie eine Liste von Text-Optionen in der DesignerBar anzeigt. Die Syntax ist wie folgt:

`input $<var name> (<"Semikolon getrennte Optionen", < initialer Wert>);`

Beispiel:

```
input $options ("option0;option1;option2", 1);
```

Dies erscheint in der DesignerBar wie folgt:



Intern werden die Optionen beginnend mit 0 nummeriert. Das obige Beispiel ist äquivalent zu folgender Definition ohne Verwendung von Textoptionen:

```
input $options (0, 2, 1);
```

Gleitkomma-Variablen

In den meisten Fällen benötigen die input-Variablen ganzzahlige Wert. Falls eine input-Variable float-Werte aufnehmen soll, benötigt ihre Deklaration zwei weitere Angaben: die Anzahl der zu verwendenden Nachkommastellen (Präzision) und die *Schrittweite*. Die Schrittweite gibt an, um wieviel NanoTrader im Rahmen der Optimierung die Variable mindestens verändern soll.

Die Deklaration einer input-Variablen für float-Wert ist wie folgt:

`input $<var Name> (<min Wert>, <max Wert>, <initialer Wert >, <Schrittweite>, <Präzision>,);`

Beispiel:

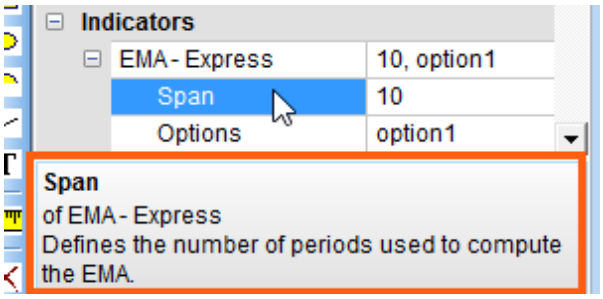
```
input $faktor (1.00, 3.00, 2.00, 0.01, 2);
```

Erläuterungen anzeigen

Eine input-Variable kann um eine `info`-Angabe ergänzt werden. Beispiel:

```
input $span (1, 100, 20) info "Defines the number of
   periods used to compute the EMA.";
```

Bei Anklicken des Parameters in der DesignerBar wird der Text in dem Erklärungsfeld angezeigt:



Dies ermöglicht die Angabe kurzer Erklärungstgexte, so dass der Anwender eines Express-Skripts nicht auf externe Dokumentation zurückgreifen muß.

7.6 Zugriff auf Variablen und Elemente aus Serien

Der Wert einer Variablen repräsentiert ihren gegenwärtigen Wert, z.B.

```
factor * delta
```

wobei `factor` und `delta` zwei numerische Variablen sind.

Ein Element aus einer `series`-Variablen wird immer *relativ* zur aktuell bearbeiteten Periode referenziert. Dabei wird die folgende Syntax verwendet:

<series name>`[`*n*`]`

wobei *n* eine positive ganze Zahl ist.

Beispiel:

`close[1]` oder `close[span]`, wobei `span` eine numerische Variable ist.

Mit `close[1]` wird also der Schlusskurs der vorigen Periode referenziert.

Um auf das Element der aktuell bearbeiteten Periode zuzugreifen, genügt es, den Namen der Serie zu verwenden, d.h. `close` ist gleichbedeutend mit `close[0]`.

7.7 Zugriff auf Array-Elemente

Ein Array hat eine feste Anzahl Elemente des Typs `numeric`. Ein Array wird in der `Variable Declarations Section` wie folgt definiert:

```
Array a[100];
```

Dadurch wird ein Array mit 100 Elementen definiert.

Der Zugriff auf Array-Elemente nutzt die gleiche Syntax wie beim Zugriff auf Elemente innerhalb einer Series:

<array name>`[n]`

Beispielsweise wird durch `a[0]` auf den ersten Array-Eintrag zugegriffen.

Beachten Sie, dass die Indizierung eines Array immer bei 0 beginnt, d.h. die gültigen Indizes für das Array `a[100]` sind 0 bis 99.

Beachten Sie ebenfalls folgenden fundamentalen Unterschied bei der Adressieung von Array-Elementen gegenüber der Adressierung von Series-Elementen: Bei einer Array-Adressierung bezeichnet der Index immer einen absoluten Eintrag innerhalb des Arrays - und nicht wie bei Serien einen relativen Versatz gegenüber dem Index der aktuellen Periode.

Die Größe eines Arrays kann mittels der Funktion SetArraySize() dynamisch geändert werden, z.B. SetArraySize(a, 250);

Alle Elemente eines Arrays können mittels SetArrayTo() auf einen Wert initialisiert werden, z.B. SetArrayTo(a, 500);

7.8 Zuweisungen

Mithilfe des `=` Operators kann einer Variablen ein Wert zugewiesen werden, z.B.

```
span = high - low;
```

Bei Zuweisung eines Wertes an eine `series`-Variable verwendet Express immer das Element der Serie, das der aktuelle bearbeiteten Periode in der `calculation section` entspricht.

Angenommen `median` sei eine `series`-Variable und Express führt die Berechnungen gerade für die 25. Periode durch, dann wird durch die Zuweisung

```
median = (high + low + close) / 3;
```

dem 25. Element der `median` Serie das Ergebnis des numerischen Ausdrucks zugewiesen.

7.9 Zuweisung von Sentimenten

Falls nicht eines der vordefinierten Interpretations-Schemata verwendet wird, müssen die berechneten Sentiment-Werte der vordefinierten Serie `sentiment` zugewiesen werden. Ein Sentiment-Wert muss zwischen 0 und 100 liegen. Falls ein Wert kleiner 0 zugewiesen wird, verwendet Express automatisch 0 und falls ein Wert größer als 100 zugewiesen wird, den Wert 100.

Bei der Festlegung von Sentimenten ist es manchmal erwünscht, das Sentiment nicht nur für die aktuelle Periode festzulegen, sondern auch für die folgenden Perioden. Dadurch kann ein wichtiges Ereignis über eine längere Zeit Einfluss auf die Gesamtstimmung nehmen.

Express ermöglicht dies durch folgende Notation:

```
sentiment = [<wert>; ...; <wert>;];
```

Beispiel:

```
sentiment = [100; 90; 80;];
```

Durch diese Zuweisung erhält die aktuelle Periode den Stimmungswert 100, die nächste den Wert 90 und die übernächste den Wert 80.

Diese sogenannte *Listen-Zuweisung* ist nur für die vordefinierte Serie `sentiment` zulässig.

7.10 Vordefinierte Serien

Die folgenden Serien zum Zugriff auf die Daten des zu analysierenden MasterCharts sind vordefiniert:

`open, close, high, low, volume`. Es können auch die Kurzschreibweisen `o, c, h, l, v` verwendet werden.

Für Renko-Charts stehen die Serien `renkoHigh, renkoLow` zur Verfügung, um auf das Hoch bzw. Tief der jeweiligen Box zuzugreifen.

Die folgenden Serien vom Typ `time` sind ebenfalls vordefiniert:

Series	**Bedeutung**
`date`	das Datum des Perioden-Endes
`dateOpen`	das Datum des Perioden-Beginns
`time`	die Uhrzeit des Perioden-Endes
`timeOpen`	die Uhrzeit des Perioden-Beginns
`dateTime`	Datum und Uhrzeit des Perioden-Endes
`dateTimeOpen`	Datum und Uhrzeit des Perioden-Beginns

Schließlich muss die vordefinierte Serie `sentiment` verwendet werden, um die berechneten Stimmungswerte zu speichern, falls nicht eines der standardisierten Interpretations-Schemata verwendet wird.

Hinweis: Mit Ausnahme der Serie `sentiment` sind alle vordefinierten Serien nur lesbar, d.h. es können ihnen keine Werte zugewiesen werden.

7.11 Importieren einer Series aus einem Sentimentor

Eine `series`-Variable kann an eine `series` aus einem anderen Express-Sentimentor oder einem Standard-Sentimentor gebunden werden, der Bestandteil der Studie ist. Die Syntax ist wie folgt:

```
series myseries (sentimentorName.seriesName);
```

Die Variable `myseries` wird eine read-only-Referenz auf `seriesName` des Sentimentors mit der id `sentimentorName`.

Dieses Schema ist besonders dann hilfreich, wenn man die Berechnungen eines Sentimentors, der zur Sentiment-Bestimmung eingesetzt wird, in einem Stop verwenden möchte. Das folgende Beispiel demonstriert dies:

```
Express EMA_Mid
Vars
series emaHigh, emaLow, emaMid;
input $span (0, 200, 10);
Calculation
if IsFirstBar() then
begin
  ExpMovingAverage(high, emaHigh, $span);
  ExpMovingAverage(low, emaLow, $span);
end
emaMid = (emaHigh + emaLow) / 2;
interpretation TriggerLine(close, emaMid);
plot (emaMid, "blue", 2);
```

Der EMA_Mid-Sentimentor ermittelt Signale basierend auf dem EMA der Periodenhochs und -Tiefs. Nehmen wir an, ein Stop soll auf genau dieser Berechnungsgrundlage aufsetzen. Anstatt nun den Code in dem Stop zu replizieren und sicherzustellen, dass die Parameter stets gleich gesetzt sind, kann die Serie `emaMid` einfach importiert werden:

```
Express Stop EMA_MidStop
vars
input $ofs (-25, 25, 10);
series anchor (EMAMidExpress.emaMid);
//The stop trails along the imported series with $ofs
//ticks.
numeric last;
calculation
if MarketPosition() = 1 then
begin
  if IsIntradayEntry() then
    last = -999999;
  last = max (last, anchor - $ofs * TickSize());
  SetStopPrice(last);
end
else
begin
  if IsIntradayEntry() then
```

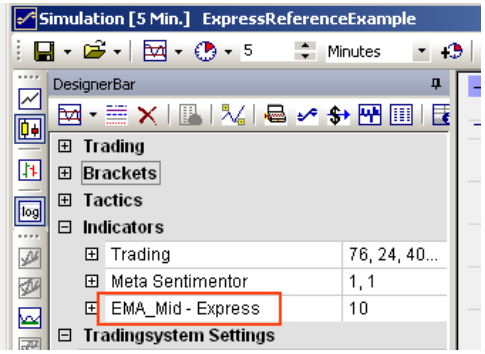
```
    last = 999999;
  last = min (last, anchor + $ofs * TickSize());
  SetStopPrice (last);
end
```

Namens Konvention

Die Bezeichnung eines Sentimentors entspricht der Benennung, wie sie in der DesignerLeiste verwendet wird, wobei alle nicht-alphabetischen Zeichen sowie Leerzeichen entfernt werden.

Der Sentimentor „EMA_Mid – Express“ aus dem Bild rechts hat daher die Bezeichnung `EMAMidExpress`.

Die Groß/Kleinschreibung wird ignoriert, d.h. `EmamidExpress` ist ebenfalls eine gültige Bezeichnung.



Wenn in der Studie mehrere „EMA_Mid – Express“-Sentimentoren vorhanden sind, muss die Bezeichnung indiziert werden:

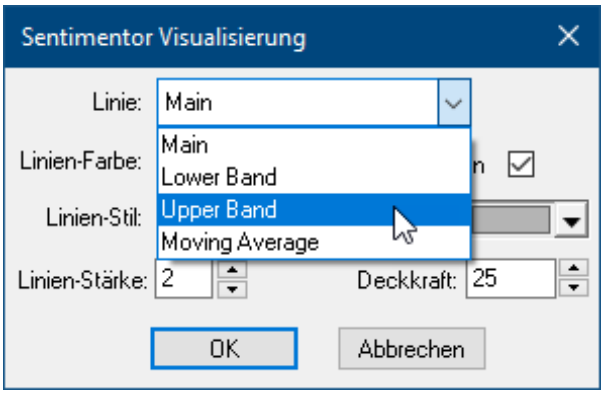
```
series anchor (EMAMidExpress2.emaMid);
```

Import von einen Standard-Sentimentor

Es ist auch möglich, Serien aus einem Standard-Sentimentor zu importieren, z.B.:

```
series myBollinger (BollingerBands.upperBand);
```

Die verfügbaren Serien werden im Dialog Sentimentor Visualisierung angezeigt. Für die Bollinger Bands stehen folgende Serien zur Verfügung:



Aufrufreihenfolge

Wenn Sentimentor A eine Serie aus Sentimentor B importiert, dann muss A neu berechnet werden sobald sich die Einstellungen für B ändern. NanoTrader stellt sicher, dass die Aufrufsequenz der Sentimentoren stets korrekt ist.

Ein Sentimentor kann beliebig viele Serien importieren. Desweiteren können die Imports geschachtelt werden, d.h. A kann von B importieren , B von C usw.

7.12 Importieren eines Arrays aus einem Sentimentor

Auch ein Array kann aus einem Sentimentor importiert werden.

Beispiel:

```
Express Array_Exporter
Vars
numeric i;
array levels[500];
Calculation
  ...
  levels[i] = close – open;
  …
```

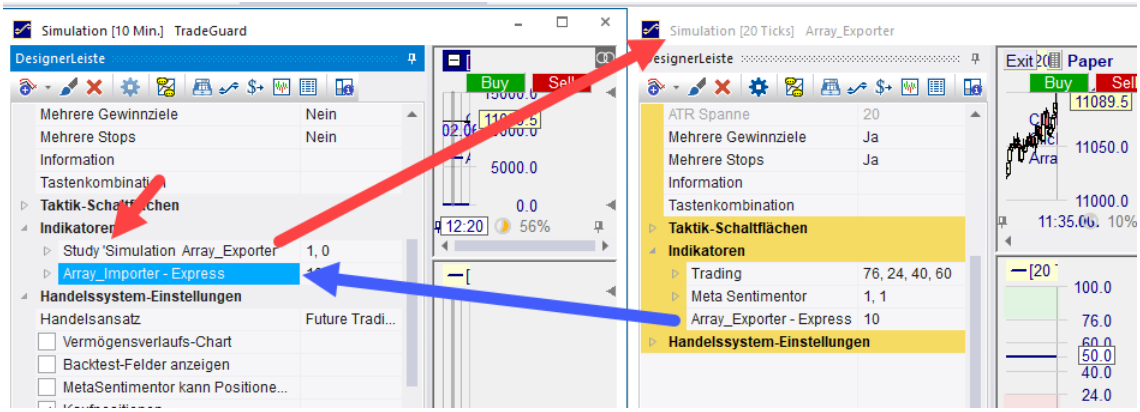
Das Array `levels` kann unter Beachtung der Namens

```
Express Array_Importer
Vars
array importedArray[ArrayExporterExpress.levels];
series test;
Calculation
  if CurrentBarIndex() < GetArraySize(importedArray) then
     test = importedArray[CurrentBarIndex()];

interpretation begin end
plot (test, primary, 2);
```

Mittels des Study-Sentimentors kann sogar ein Array aus einem Sentimentor importiert werden, der sich in einer anderen Studie befindet:

```
array importedArray[study.ArrayExporterExpress.levels];
```

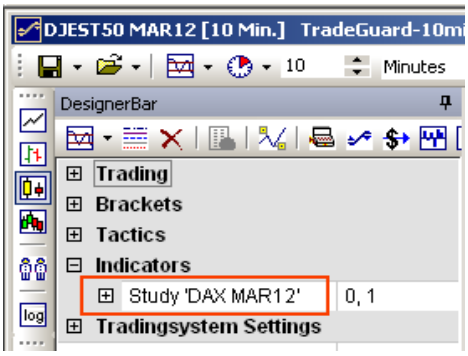


Ein importiertes Array ist schreibgeschützt, d.h die Elemente das Arrays können nur gelesen, aber nicht geschrieben werden.

7.13 Importieren der Preisdaten eines anderen Symbols

Als Spezialfall des oben beschriebenen Importierens einer Serie aus einem anderen Sentimentor kann auch auf die Preisdaten eines durch den Study-Sentimentor verwendeten Symbols zugegriffen werden.

Nehmen wir an, ein Study-Sentimentor greift auf den DAX-Future zu:



Ein Express-Sentimentor kann die Preisdaten nun wie folgt importieren:

```
series dax (StudyDaxMar.close); //or .open, .high, .low
```

Future-Kontrakte haben üblicherweise ihr Verfallsdatum am Ende ihres Symbolnamens angefügt. Das führt dazu, dass bei einem Wechsel des Kontrakts der Express-Code angepasst werden müsste. Um dies zu vermeiden, erlaubt Express auch die Angabe lediglich der Anfangssequenz des Namens:

```
series dax (StudyDax.close);
```

7.14 Datum-und Zeit-Konstanten

Eine Zeit-Konstante hat das folgende Format:

`HH:MM` oder `HH:MM:SS`

Beispiel:

```
14:15    14:50:40
```

Falls die Sekunden weggelassen werden, werden diese automatisch auf 0 gesetzt.

Das Format eines Datums lautet:

```
DD_MM_YYYY
```

Beispiel:
22_10_2002

Datum-und Zeit-Konstanten können in booleschen Ausdrücken verwendet werden, z.B.

```
//nicht in der ersten Handelsstunde kaufen
if time < 10:00 then
  sentiment = 50;
else …
```

7.15 Anweisungen

Eine Anweisung ist eine Zuweisung oder ein Prozeduraufruf, jeweils abgeschlossen durch eine Semikolon, oder eine komplette Kontrollstruktur.

Beispiele für Anweisungen:

```
delta = high – low;
MovingAverage (close, mySeries, 10);
if (open > close) then up = up + 1;
```

7.16 Kontroll-Strukturen

7.16.1 if

Die `if` -Kontrollstruktur wird verwendet, falls gewisse Anweisungen nur ausgeführt werden sollen, falls eine Bedingung erfüllt ist. Die Syntax lautet:

```
if <boolescher Ausdruck> then
   <Anweisung>;
```

Falls mehrere Anweisungen ausgeführt werden sollen, müssen diese mittels begin und end zu einem *Anweisungsblock* zusammengefasst werden:

```
if < boolescher Ausdruck > then
begin
  < Anweisung >;
  < Anweisung >;
  ...
  < Anweisung >;
end
```

7.16.2 If then else

Eine `if` –Kontrollstruktur kann auch Anweisungen enthalten, die ausgeführt werden sollen, falls die Bedingung *nicht* erfüllt ist:

```
if <boolescher Ausdruck> then
   <Anweisung>;
```

```
else
   < Anweisung >;
```

Auch hier können mehrere Anweisungen zu einem Block zusammengefasst werden:

Beispiel:

```
if close > open then
  upMoves = upMoves + 1;
else
begin
  downMoves = downMoves + 1;
  downVol = downVol + volume;
end;
```

7.16.3 While Schleife

Die Syntax einer `While` Schleife lautet:

```
while < boolescher Ausdruck >
begin
  < Anweisung >;
  < Anweisung >;
...
  < Anweisung >;
end
```

Die Anweisungen werden solange ausgeführt, bis der <boolesche Ausdruck> false ergibt.

Beispiel:

```
lastHigh = 1;
vol = 0;
while (close[lastHigh] <> void)
       and (close > close[lastHigh])
begin
  vol = vol + volume[lastHigh];
  lastHigh = lastHigh + 1;
end
```

Hinweis: Es muss stets sehr sorgfältig sichergestellt werden, dass der boolesche Ausdruck tatsächlich schließlich `false` ergibt - ansonsten würde die Schleife endlos laufen. Um dieses zu verhindern, beendet NanoTrader das Express-Programm mit einer entsprechenden Fehlermeldung automatisch, falls die Schleife nicht innerhalb von 5 Sekunden terminiert.

7.16.4 For Schleife

Die Syntax der `for`-Schleife lautet:

```
for <variable> = <start wert> to <numerischer Ausdruck>
begin
```

```
  <Anweisung>;
  < Anweisung >;
...
  < Anweisung >;
end
```

Zu Beginn der `For`-Schleife wird die Variable <variable> auf den <start wert> gesetzt. Falls <variable> den Wert des < numerischen Ausdrucks> nicht überschreitet, werden die Anweisungen ausgeführt und anschließen der Wert von <variable> um eins erhöht. Dieser Prozess wird solange wiederholt, bis <variable> schließlich den Wert <numerischer Ausdruck> überschreitet.

Beispiel:

```
upMoves = 0;
for i = 0 to 9
begin
  if close[i] > open[i] then
    upMoves = upMoves + 1;
end
```

Manchmal ist es erforderlich, den Wert von <variable> zu vermindern und die Schleife zu beenden, wenn <variable> den Wert des <numerischen Ausdrucks> unterschreitet. Dies kann durch folgende Variante der `For`-Schleife erreicht werden:

```
for <variable> = <start wert> downto <numerischer Ausdruck>
begin
  <Anweisung>;
  < Anweisung >;
...
  < Anweisung >;
end
```

Hinweis: Es muss stets sehr sorgfältig sichergestellt werden, dass der Wert von <variable> den Wert des numerischen Ausdrucks> schließlich überschreitet bzw. unterschreitet im Falle der `downto`-Variante - ansonsten würde die Schleife endlos laufen. Um dieses zu verhindern, beendet NanoTrader das Express-Programm mit einer entsprechenden Fehlermeldung automatisch, falls die Schleife nicht innerhalb von 5 Sekunden terminiert.

7.17 Performanz bei der Berechnung von Sentimentoren

Immer wenn `while` oder `for`-Schleifen verwendet werden, ist es wichtig darauf zu achten, nicht unnötig viel Rechenzeit in Anspruch zu nehmen. Mit anderen Worten, es muss darauf geachtet werden, dass die Berechnungen *performant* durchgeführt werden.

Es ist sehr einfach, durch eine „naive“ Implementierung einen Sentimentor zu erstellen, der zwar die gewünschten Ergebnisse berechnet, aber dessen

Implementierung zu einer drastischen Verlangsamung des Gesamtsystems führt. Dazu ein Beispiel:

Angenommen es sollte ein gleitender 50-Perioden-Durchschnitt berechnet werden. Ein naiver Ansatz würde einfach die Summe des gegenwärtigen Kurses und der vorigen 49 Kurse bilden und durch 50 dividieren. Diese Operationen würde dann für jede Periode ausgeführt werden und das Ergebnis wäre korrekt. Eine etwas intelligentere Implementierung würde die Tatsache ausnutzen, dass der jeweils neu zu berechnende Durchschnittwert sich aus dem vorigen berechnen lässt, indem der am weitesten „links“ liegende Preis „herausgerechnet“ und der aktuelle Preis „hineingerechnet“ wird.

Bei genauerer Betrachtung stellt man fest, dass der „naive“ Ansatz 50 mal soviel Rechenaufwand verursacht, wie der „intelligente“. Bei Berechnung eines 200-Perioden-Durchschnitts wäre der Aufwand sogar 200 mal so hoch. Man kann sich leicht vorstellen, welche Auswirkungen dies auf die Geschwindigkeit bei einer Optimierung hat...

Sehr häufig können manche Berechnungen durchgeführt werden, nachdem alle Perioden abgearbeitet wurden, beispielsweise wenn ein Gleitender Durchschnitt auf die Werte einer berechneten Serie ermittelt werden soll. Dies kann leicht unter Verwendung der booleschen Funktion `IsFinalBar()` realisiert werden:

```
...
series result;
input $span (1, 200, 10);
Calculation:
  result = ...;
  if IsFinalBar() then //true, falls letzte Periode
     MovingAverage (result, result, $span);
```

`I`mmer wenn Sie davon ausgehen, dass Ihr Express-Skript viel Rechenzeit benötigt, stellen Sie sicher, dass Sie `CalculateAtEveryTick(false);` am Anfang des Skripts aufrufen. Dadurch wird sichergestellt, dass das Skript nur *am Ende einer Periode* ausgeführt wird und nicht bei jedem eingehenden Tick. Dies verringert die Gesamtarbeitslast natürlich enorm.

Hinweis: Nutzen Sie regelmäßig die Code-Analyse, indem Sie auf das Symbol in der Toolbar klicken. Dadurch werden Sie auf einige offensichtliche Ineffizienzen hingewiesen, die Ihr Code möglicherweise enthält, sowie auf nicht verwendete Variablen und Serien.

7.18 Interpretation – Bestimmung der Sentimente

Der Hauptgegenstand eines Sentimentors ist natürlich die Berechnung von Sentimenten. Die entsprechenden Anweisungen werden in der `Interpretation Section` aufgeführt. Das Schlüsselwort `Interpretation` leitet diese ein.

7.18.1 Verwendung der standardisierten Interpretations-Schemata

Sehr häufig kann die Berechnung der Sentimente mithilfe eines der standardisierten Interpretations-Schemata durchgeführt werden, welche auch von den direkt in NanoTrader verfügbaren Sentimentoren eingesetzt werden. Wenn ein solches Schema verwendet wird, kann auch der zugehörige Dialog zum Konfigurieren der Interpretation verwendet werden. Die Details hierzu sind in der NanoTrader *Bedienungsanleitung* beschrieben.

Der Rückgriff auf eines der Schemata erleichtert also die Erstellung eines Sentimentors ganz erheblich.

Ein Schema wird wie eine normale Express-Funktion aufgerufen.

Beispiel:

```
interpretation TwoThresholds (mySeries, $upZone,
                              $downZone);
oder
interpretation TriggerLine (close, mySeries);
```

Manche Interpretations-Schemata benötigen neben der oder den zu betrachtenden Serien noch `input`-Variablen als weitere Parameter.

Die folgenden Schemata stehen zur Verfügung

Schema	**Typische Verwendung**

`TwoThresholds (series curve,`
`  input upThreshod,`
`  input downThreshold)`
	RSI

`TriggerLine (series curve,` `  series trigger)`	Crossing MA

`Swing (series curve`
`  input spanLeft,`
`  input spanRight)`
	Momentum

`Bands (series curve,`
`  series lower,`
`  series upper)`
	Bollinger Bands

Falls ein Interpretations-Schema verwendet wird, kann auf die Angabe von plot-Anweisungen verzichtet werden – in diesem Fall würde Express die von dem Schema verwendeten Serien automatisch zeichnen.

7.18.2 Explizite Programmierung der Interpretation

Falls kein standardisiertes Interpretations-Schema eingesetzt werden kann, müssen die Sentimente explizit in der `interpretation section` vergeben werden:

```
interpretation
begin
  <statement>
…
  <statement>
```

```
end
```

Man beachte die `begin` und `end` Anweisungen, die die Anweisungen der `interpretation section` zu einem Block zusammenfassen.

Genau wie in der `calculation section` werden die Perioden iterativ von „links nach rechts“ abgearbeitet. Die Sentimente müssen der vordefinierten Serie `sentiment` zugewiesen werden. NanoTrader initialisiert diese Serie automatisch mit dem neutralen Stimmungswert 50.

Beispiel:

```
interpretation
begin
  if CrossesAbove (close, mySeries) then
     sentiment = 100;
end
```

Bei der Zuweisung von Sentimenten kann die sogenannte *Listen Zuweisung* verwendet werden, um bei wichtigen Ereignissen auch für die folgenden Perioden einen Stimmungswert festzulegen.

Beispiel:

```
interpretation
begin
  if CrossesAbove (close, mySeries) then
     sentiment = [100; 90; 80;];
  else if close > mySeries then    //staying above mySeries
    if sentiment = 50 then         //do not overwrite
                                   //crossing event
      sentiment = 65;
end
```

7.19 Zeichnen

Ein Express-Programm wird durch ein oder mehrere Plot-Anweisungen abgeschlossen. Die Syntax lautet:

```
plot (<series name>, <Farbe>, <Strichbreite>);
```

oder

```
plotline (<Wert>, < Farbe >, < Strichbreite>);
```

Beispiel:

```
plot (mySeries, “blue”, 2);
plotline ($threshold, “green”, 1);
```

Manchmal ist es interessant, einen Kerzen-Chart oder Bar-Chart mit modifizierten Preisdaten zu zeichnen. Dazu stehen die folgenden Plot-Routinen zur Verfügung:

`plotcandles (<open series>, <close series>, <high series>, <low series>);`

und

`plotbars (<open series>, <close series>, <high series>, <low series>);`

Um den Bereich zwischen zwei Kurven auszufüllen steht die plotband-Funktion zur Verfügung (z.B. für Sentimentoren ähnlich Bollinger Bands):

```
plotband (<oberer series Name>, < Farbe >, < Strichbreite >,
          <unterer series Name>, < Farbe >, < Strichbreite >,
          <Füllfarbe>);
```

Um zwei sich kreuzende Serien zu zeichnen und die eingeschlossenen Flächen in entsprechenden Farben zu füllen, kann die Funktion `plotcrossinglines` eingesetzt werden (z.B. für Sentimentoren ähnlich Crossing Moving Averages):

```
plotcrossinglines (<series1 Name >, < Farbe >, < Strichbreite >,
                   <series2 Name >, < Farbe >, < Strichbreite >,
                   < Füllfarbe series1 über series2>,
                   < Füllfarbe series1 unter series2>);
```

7.20 Mehr zum Thema Farben

Die Farbe jedes gezeichneten Elements muss mit der Farbe des Charthintergrunds kompatibel sein. Beispielsweise ist eine dunkelblaue Linie auf schwarzem Hintergrund ist nicht sehr hilfreich. Mit anderen Worten: Jede Farbdefinition sollte sowohl auf einem hellen Chart-Hintergrund als auch auf einem dunklen Hintergrund "gut" aussehen.

Um dies zu ermöglichen, erlaubt NanoTrader die Angabe sogenannter *dualer Farben*.

7.20.1 Duale Farben

Eine duale Farbe besteht immer aus der Farbangabe für einen hellen Chart-Hintergrund, optional gefolgt von einer Farbangabe für einen dunklen Chart-Hintergrund. Das Tilde-Zeichen ~ trennt beide Angaben:

`"DarkBlue~LightGreen"`

NanoTrader wählt je nach aktuellem Hintergrund automatisch die richtige Farbe für das Plotten eines Elements, d.h. DarkBlue bei hellem Hintergrund und LightGreen bei dunklem.

Somit können Sie z.B. das Aussehen von NanoTrader von "Silber" auf "Nacht" ändern, und die Express-Sentimentoren passen sich automatisch an. Dies ist insbesondere auch dann sehr hilfreich, wenn Sie Ihre Express-Sentimentoren anderen Benutzern zur Verfügung stellen wollen.

Wenn die Farbe, die für einen dunklen Hintergrund angewendet werden soll, in der Farbspezifikation weggelassen wird, dann erstellt NanoTrader sie automatisch, indem er die Leuchtkraft der Farbe für den hellen Hintergrund als Ausgangspunkt verwendet. Aus einem "Dunkelblau auf hellem Hintergrund" wird dann ein "Hellblau auf dunklem Hintergrund".

Die einfache Angabe einer einzelnen Farbe definiert somit implizit ein duales Paar von Farben. Mit anderen Worten, duale Farben erlauben es Ihnen, das Aussehen der geplotteten Elemente genau zu spezifizieren, aber wenn Sie nur eine einzelne Farbe angeben, dann wird NanoTrader hinter den Kulissen sein Bestes tun, um einen gut aussehenden Plot auf jedem Hintergrund zu erstellen.

7.20.2 Syntax der Definition dualer Farben

Dass allgemeine Format für die Definition dualer Farben ist wie folgt:

```
<Farb-Spez. für hellen HG >~<Farb-Spez. für dunklen HG>
```

Der zweite Teil, beginnend mit `~`, ist optional.

Eine Farb-Spezifikation besteht aus dem Namen der Farbe oder aus der entsprechenden RGB-Notation. Die Rot/Grün/Blau-Werte sind ganze Zahlen zwischen 0 und 255, die den jeweiligen Farbanteil festlegen.

Beispiele:

```
plot (close, "black", 1);
```

Verwendet Schwarz auf einem hellen Chart-Hintergrund und die automatisch angepasste Farbe auf einem dunklen Chart-Hintergrund.

```
plot (close, "black~white", 1);
```

Verwendet Schwarz auf einem hellen Hintergrund und Weiß auf einem dunklen.

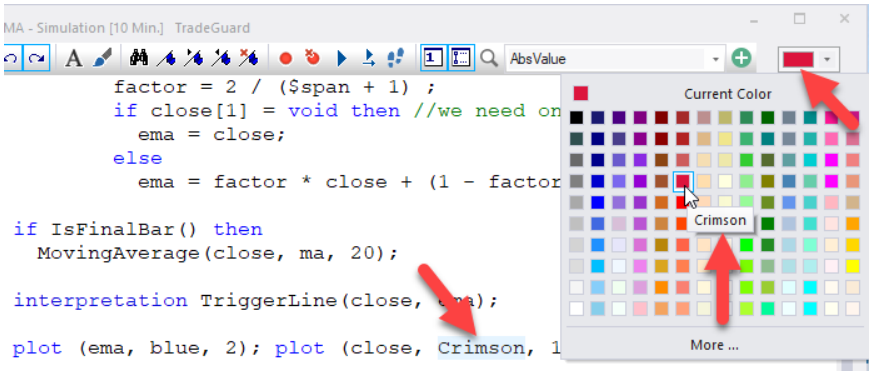
```
plot (close, "128,128,255~yellow”, 1);
```

Farbangabe für hellen Hintergrund in RGB-Notation.

```
plot (close, "1325324~0,0,255", 1);
```

Verwendet den Integer-Wert der RGB-Notation für den hellen Hintergrund.

Ein gültiger Farbname kann am einfachsten über den Farbwähler in der Toolbar in den Code eingefügt werden:



7.20.3 Syntax der Definition dualer Farben

Zusätzlich zu den Farbnamen kann ein Satz von *logischen Farben* verwendet werden. Logische Farben sind bereits für helle und dunkle Hintergründe vordefiniert. Sie können bequem verwendet werden, um auf gut aussehende Farben für typische Anwendungsfälle zurückzugreifen. Die Verwendung der gleichen Farben in ähnlichen Szenarien erleichtert das "Lesen" und Verstehen von Senti-mentor-Grafiken.

Die logischen Farbnamen lauten:

- `Primary` Die primäre Serie in einem Chart
- `Secondary` Die sekundäre Serie, bspw. bei CrossingMA
- `UpperBand`
- `LowerBand`
- `FillBand` Die Standard Füll-Farbe
- `UpperThreshold`
- `LowerThreshold`
- `Threshold`

Beispiele:

```
plot (close, "primary", 1);
plot (ma, "secondary", 1);
```

Hinweis: Eine logische Farbe definiert implizit Farben für einen hellen und dunklen Hintergrund. Daher ist eine Angabe wie "primary~red" ungültig und wird vom Express-Compiler nicht akzeptiert.

7.20.4 Angabe der Deckkraft für Füllfarben, Highlight() und Annotate()

Eine Farbangabe ermöglicht auch die Definition der Deckkraft, wenn eine Füllfarbe oder eine Farbe für die Highlight()- oder Annotate()-Funktion angegeben wird.

Die Deckkraft (auch "Alpha" genannt) ist ein Wert zwischen 0 und 100, wobei 100 bedeutet, dass die Farbe vollständig deckend ist. (Hinweis: Verwechseln Sie dies nicht mit RGB-Werten, die im Bereich von 0...255 liegen).

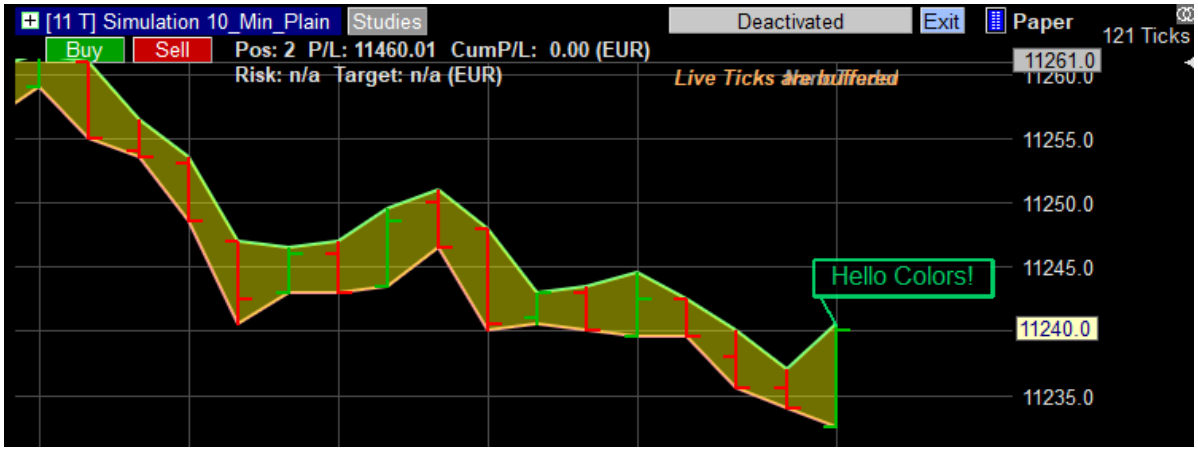
Die Deckkraft ist optional. Wenn sie nicht angegeben wird, ist sie standardmäßig auf 25 eingestellt. Ansonsten ist wird sie durch ein Komma von der Farbangabe getrennt.

Beispiele für Farbspezifikationen mit Deckkraft:

```
"red,7"
"primary,75"
"128,128,22,75"
"red,75~blue,60"
```

Anwendungs-Beispiele:

```
Highlight("textAbove:Hello Colors!", "blue,80~SpringGreen,80");
plotband(high, "green", 2, low, "red", 2, "blue,40~yellow,44");
```



8 Ein Beispiel-Blocker

Zusätzlich zu den Stimmungswerten der Skala 0 bis 100 unterstützt NanoTrader zwei spezielle Stimmungszustände, die im Zusammenhang mit Filtern genutzt werden:

- BLOCK
  Sowohl Long als auch Short-Signale werden abgewiesen
- FLAT
  Sowohl Long als auch Short-Signale werden abgewiesen. Desweiteren wird eine evtl. offene Position geschlossen.

NanoTrader erlaubt bei der Definition Manueller Sentimentoren die Verwendung dieser Zustände, z.B. um zu bestimmten Tageszeiten den Handel zu unterbinden.

Die Programmierung eines solchen Blockers in Express illustriert das folgende Beispiel:

`Express Blocker VolCheck` `Vars`	`1.`

```
series twoPeriodVol;

Calculation
if CurrentBarIndex () > 1 then
  twoPeriodVol = vol[1] + vol;

interpretation
begin
  if twoPeriodVol < 10000 then
    sentiment = senti_block;                    2.
  else if twoPeriodVol > 5000000 then
    sentiment = senti_flat;                     3.
  else
    sentiment = senti_pass;                     4.
end

plot (twoPeriodVol, blue, 2);
```

Erläuterung:

1. Das Schlüsselwort `Blocker` zeigt an, dass es sich bei dem nachfolgenden Code um einen Blocker handelt. Dadurch können die Sentiment-Konstanten `senti_block`, `senti_flat` und `senti_pass` verwendet werden. Desweiteren wird dadurch erzwungen, dass dieser Sentimentor nur als Filter in eine Analyse eingefügt werden kann.
2. Die Konstante `senti_block` sorgt dafür, dass sowohl Long als auch Short-Signale abgewiesen werden.
3. Die Konstante `senti_flat` sorgt dafür, dass sowohl Long als auch Short-Signale abgewiesen werden und das eine etwaige offene Position geschlossen wird.
4. Die Konstante `senti_pass` lässt alle Signale passieren.

9 Ein Beispiel-Stop

Die Berechung eines preisbasierten Stops in Express wird an folgendem Beispiel illustriert:

```
Express Stop Simple                             1.
vars
input $increase (1, 25, 10);
series ma;

calculation
  if IsFirstBar () then
    MovingAverage (close, ma, 10);
```

```
if MarketPosition() = 1 then    //long                                  2.
begin
  if IsIntradayEntry() then //we just opened the position              3.
    SetStopPrice (EntryPrice() - 15);                                   4.
  else
    SetStopPrice (ma - 100 + $increase* BarsSinceEntry());
end
else if MarketPosition() = -1 then  //short
begin
  if IsIntradayEntry() then //we just opened the position
    SetStopPrice (EntryPrice() - 15);
  else
    SetStopPrice (ma + 100 - $increase* BarsSinceEntry());
end                                                                     5.
```

Erläuterung:

1. Das Schlüsselwort `Stop` zeigt an, dass es sich bei dem nachfolgenden Code um einen preisbasierten Stop handelt. Dadurch können Funktionen aufgerufen werden, die nur für Stops verfügbar sind. Desweiteren wird dadurch erzwungen, dass dieser Sentimentor nur als Stop in eine Analyse eingefügt werden kann.
2. Die Funktion `MarketPosition()` gibt Auskunft über die aktuelle Position:
   1 = long
   0 = flat
   -1 = short
3. Die boolesche Funktion `IsIntradayEntry()` liefert `true`, falls die Position gerade in einer noch nicht abgeschlossenen Periode eröffnet wurde. In diesem Fall ist es oftmals nötig, eine andere Stoppreis-Berechnung durchzuführen, z.B. rein auf Basis des Einstiegspreises. Wird die Position nicht zum Periodenende glattgestellt, so wird nochmals ein Stopkurs berechnet, der dann für die nächste Periode verwendet wird.
4. Die Funktion `SetStopPrice()` macht den berechneten Stop bekannt. NanoTrader verwendet dann den engsten aller eingesetzten Stops.
5. Für Stop-Sentimentoren gibt es weder eine `Interpretation Section` noch eine `Plot Section`

10 Ein Beispiel-Stop/Taktik mit Intraperioden-Updates

Der folgende Stop wird mit jedem ankommenden Tick neu berechnet. Bei Erreichen eines Triggerpreises setzt er sich auf ein Breakeven-Niveau und beginnt von dort aus zu „trailen“.

```
Express Stop BETrailStop

vars
input $initialRisk(0, 50, 10);
input $profitTrigger(0, 5, 3);
input $initialProfitOffset(0, 5, 2);
input $trail(0, 10, 5);

numeric entryPrice, tickSize;
numeric extreme;
numeric breakEven, trailStop;

calculation
if IsFirstBar() then
begin
  SetIntraPeriodUpdate();                                          1.
  entryPrice = EntryPriceOriginal();                               2.
  tickSize = TickSize();
end

if MarketPosition() = 1 then //Long position
begin
  if IsIntradayEntry() then
    extreme = MaxPriceEntryBar();                                  3.
  else if (BarsSinceEntry() = 0) then//We entered via a study
                                     //at the end of the period
    extreme = close;
  else
    extreme = max (extreme, Highest(high, BarsSinceEntry()));

  breakEven = entryPrice + $profitTrigger * tickSize;
  if $trail = 0 then  //trail deactivated?
     trailStop = -9999;
  else
    trailStop = extreme - $trail * tickSize;

  if extreme >= breakEven then
     SetStopPrice (max(entryPrice + $initialProfitOffset *
                       tickSize, trailStop));
  else
     SetStopPrice(entryPrice - $initialRisk * tickSize);
end
else if MarketPosition() = -1 then  //Short position
begin
  if IsIntradayEntry() then
    extreme = MinPriceEntryBar();
  else if (BarsSinceEntry() = 0) then
    extreme = close;
  else
    extreme = min (extreme, Lowest (low, BarsSinceEntry()));
```

```
  breakEven = entryPrice - $profitTrigger * tickSize;
  if $trail = 0 then
     trailStop = 999999;
  else
    trailStop = extreme + $trail * tickSize;

  if extreme <= breakEven then
    SetStopPrice (min(entryPrice - $initialProfitOffset *
                      tickSize, trailStop));
  else
     SetStopPrice(entryPrice + $initialRisk * tickSize);
end
```

Erläuterung:

1. Die Existenz dieser Routine im Quellcode aktiviert die intraperioden-Berechnung.
2. Der Erföffnungspreis der Position laut Konto.
3. Der höchste Preis der Eröffnungsperiode, nachdem die Position eingegangen wurde. Bsp.: 60-Minuten Perioden. Positionseröffnung nach 30-Minuten => geliefert wird das Hoch der letzen 30-Minuten.

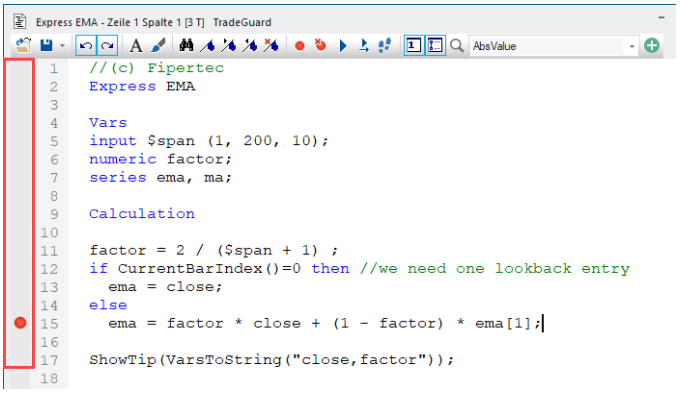
11 Debuggen von Express-Programmen

Die Express-Umgebung ist mit einem sogenannten *Debugger* ausgestattet. Ein Debugger ermöglicht es, die Codeausführung an bestimmten *Haltepunkten* anzuhalten und die Variablen und den Programmablauf auszuwerten. Dadurch lassen sich Fehler leicht erkennen und beheben.

11.1 Haltepunkte setzen

11.1.1 Im Editor

Der schnellste Weg, einen Haltepunkt zu setzen, ist ein Klick in den Marker-Bereich auf der linken Seite des Editors:

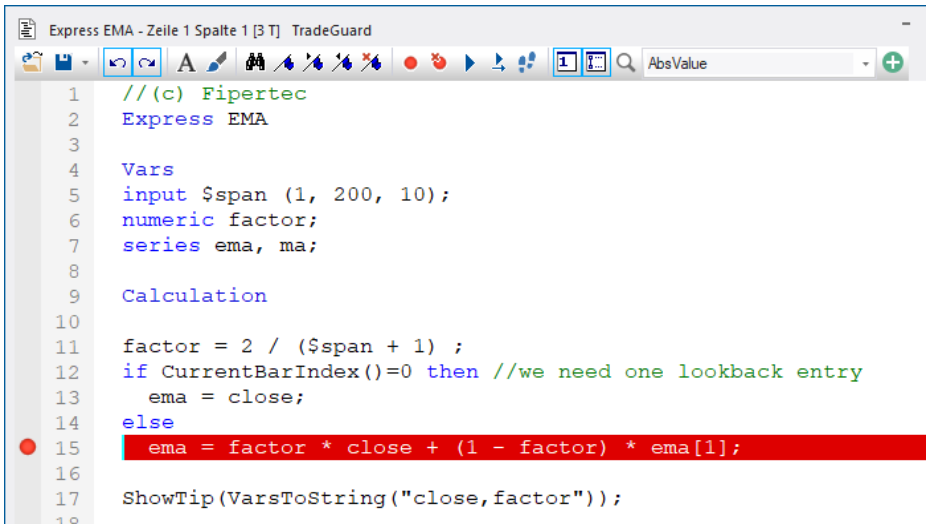


Alternativ kann ein Haltepunkt in der aktuellen Zeile über die Symbolleiste, das Kontextmenü oder durch Drücken von Strg-B gesetzt werden.

Es können beliebig viele Haltepunkte gesetzt werden.

Wenn ein Haltepunkt in einer Zeile gesetzt wird, die keinen unterbrechbaren Code enthält, wird der Haltepunkt automatisch in eine solche verschoben, sobald die Programmausführung beginnt oder fortgesetzt wird.

Sobald während der Programmausführung ein Haltepunkt auftritt, wird die Ausführung gestoppt und die Zeile, in der die Unterbrechung aufgetreten ist, wird hervorgehoben:



Die Farbe für die Hervorhebung kann im Farb-Manager eingestellt werden.

Beachten Sie, dass im obigen Beispiel die `else`-Klausel ausgeführt wurde. Ansonsten wäre der Haltepunkt ignoriert worden.

Das Programm wird angehalten, *bevor* der Code auf der Haltepunktzeile ausgeführt wird.

Haltepunkte im Editor entfernen

Um einen einzelnen Haltepunkt zu entfernen, klicken Sie ihn im Markierungsbereich an oder platzieren Sie den Cursor in der Zeile daneben und verwenden Sie die Schaltfläche in der Toolbar, das Kontextmenü oder Strg-B.

Um alle Haltepunkte zu entfernen, klicken Sie in der Toolbar oder im Kontextmenü.

11.1.2 Haltepunkte durch Funktionsaufrufe

Manchmal möchte man die Programm-Ausführung nur unter bestimmten Bedingungen unterbrechen. Um das zu erreichen, stehen die folgenden drei

Express-Funktionen zur Verfügung. Diese Funktionen sind nur aktiv, wenn der Code im Editor geöffnet ist. Andernfalls werden, werden sie ignoriert.

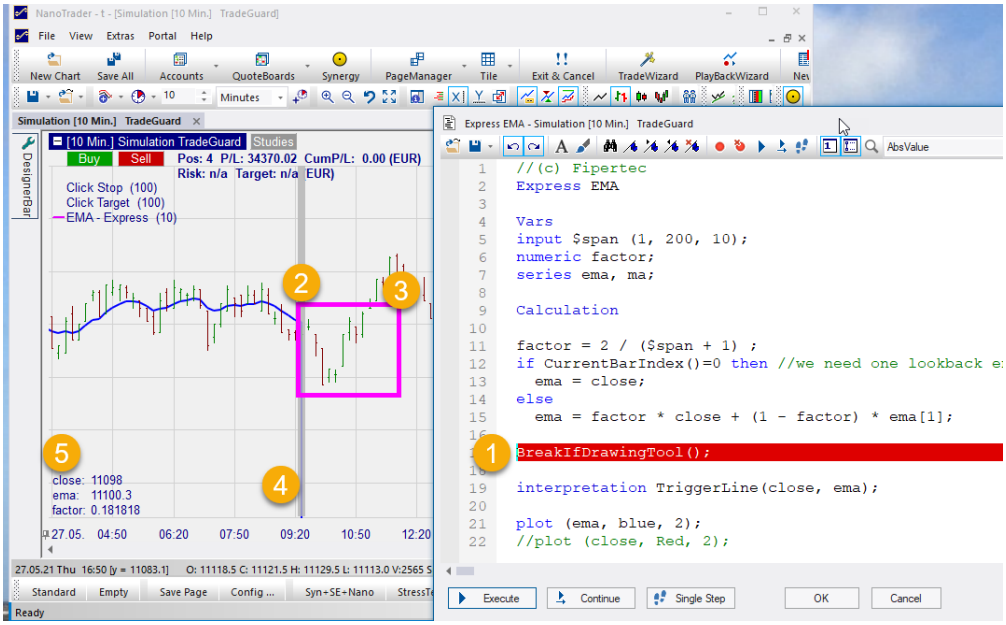
- **Break()**
  Unterbricht die Ausführung an der Stelle, wo der Funktionsaufruf geschieht:

```
9   Calculation
10
11  factor = 2 / ($span + 1) ;
12  if CurrentBarIndex()=0 then //we need one lookback entry
13    ema = close;
14  else
15    ema = factor * close + (1 - factor) * ema[1];
16  Break();
17  ShowTip(VarsToString("close,factor"));
```

- **BreakIf (boolescher Ausdruck)**
  Unterbricht die Ausführung, falls der boolesche Ausdruck `true` ergibt:

```
9   Calculation
10
11  factor = 2 / ($span + 1) ;
12  if CurrentBarIndex()=0 then //we need one lookback entry
13    ema = close;
14  else
15    ema = factor * close + (1 - factor) * ema[1];
16  BreakIf((close > open) and (CurrentBarIndex() > 5));
17  ShowTip(VarsToString("close,factor"));
18
```

- **BreakIfDrawingTool()**
  Beim Ausführen dieser Funktion prüft NanoTrader, ob *in der aktuellen Periode* im MasterChart ein Zeichenwerkzeug beginnt oder endet.
  Im untenstehenden Beispiel wurde ein Rechteck um einen "interessanten" Bereich gezeichnet. NanoTrader wird am ersten und letzten abgedeckten Balken die Programm-Ausführung unterbrechen:



(Weitere Erklärungen finden Sie im nächsten Abschnitt.)

11.2 Variablen inspizieren während eines Programm-Halts

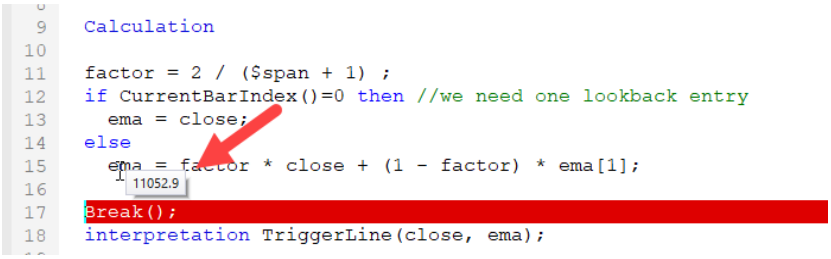
Im MasterChart

Sobald ein Haltepunkt auftritt, zeigt NanoTrader einen grauen vertikalen Balken im MasterChart an, um die Periode hervorzuheben, die gerade bearbeitet wird. Siehe 4 im obigen Beispiel. Wenn die Periode außerhalb des aktuellen Zooms liegt, wird dieser automatisch angepasst.

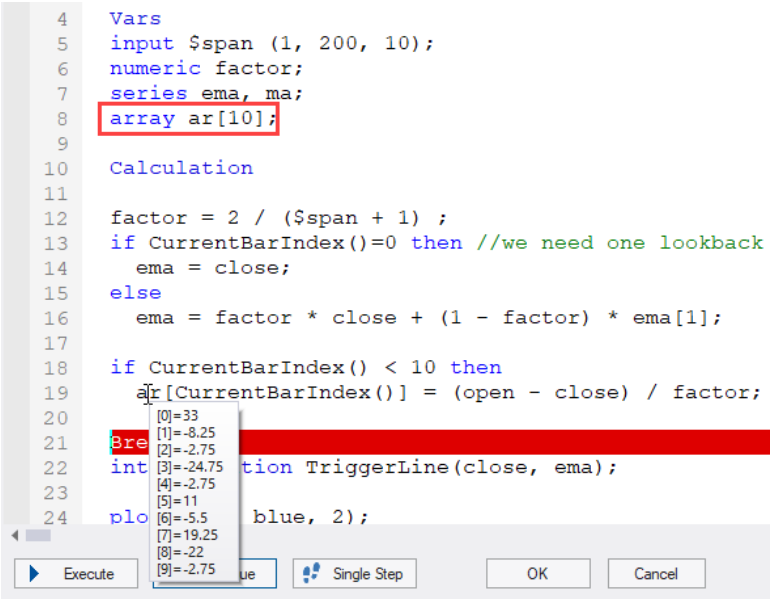
Alle im Code verwendeten Variablen sowie ihre aktuellen Werte werden im unteren Teil des MasterCharts angezeigt 5. "Verwendet" bedeutet, dass die Variable oder Serie nicht nur definiert ist, sondern auch im Code referenziert wird.

Im Editor

Während die Programmausführung angehalten ist, können Sie mit der Maus auf eine Variable oder Serie zeigen und ihr aktueller Wert wird in einem Popup-Fenster angezeigt:

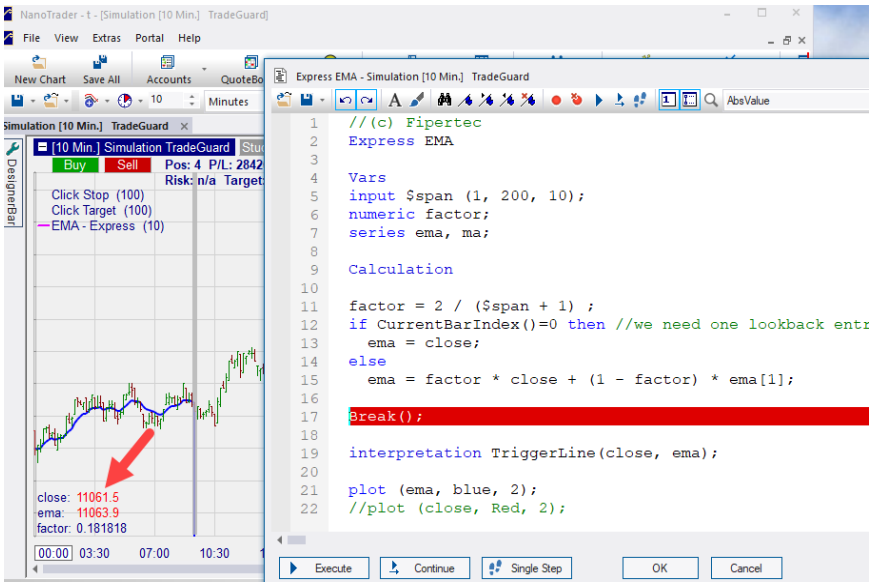


Wenn auf ein Array gezeigt wird, wird dessen gesamter Inhalt angezeigt:



11.3 Hervorhebung von geänderten Variablen

Der Wert einer Variablen wird im Chart rot dargestellt, wenn sich ihr Wert im Vergleich zum Wert beim vorherigen Haltepunkt geändert hat:



11.4 Fortsetzen der Code-Ausführung von einem Haltepunkt aus

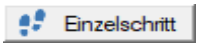
Es gibt drei Möglichkeiten mit der Code-Ausführung fortzufahren, nachdem ein Haltepunkt ausgeführt wurde:

- Ausführen

Wenn Sie auf die Schaltfläche "Ausführen" klicken, wird der Code *erneut* ausgeführt, d. h. die Ausführung beginnt mit der ersten Periode und der ersten Anweisung.
Nachdem der Code geändert wurde, sollten Sie immer "Ausführen" verwenden, um das Skript erneut auszuführen.

- Fortsetzen

Die Programmausführung wird an der Stelle fortgesetzt, wo sie angehalten wurde.

- Einzelschritt 

Die Programmausführung wird an der Stelle fortgesetzt, wo sie angehalten wurde, unterbricht aber automatich in der nächsten Zeile. Auf diese Weise lässt sich der Programmablauf leicht verfolgen, ohne dass in jeder Zeile explizit ein Haltepunkt gesetzt werden müsste. Desweiteren lassen sich so Wertänderungen der Variablen einfach nachvollziehen.

11.5 Einige Tipps & Tricks zum Debuggen

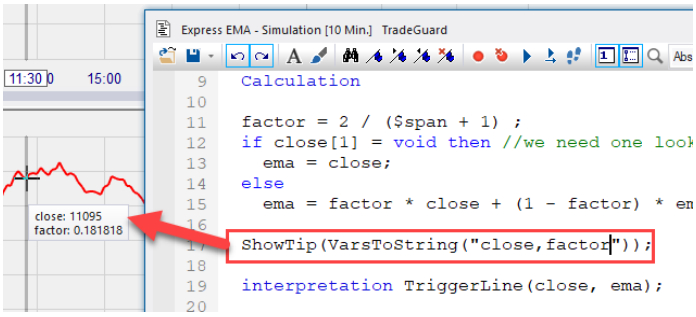
- Um die Ausführung am Anfang jeder Periode anzuhalten, setzen Sie einen Haltepunkt an der `Calculation` Klausel:

```
1  //(c) Fipertec
2  Express EMA
3
4  Vars
5  input $span (1, 200, 10);
6  numeric factor;
7  series ema, ma;
8
9  Calculation
10
11 factor = 2 / ($span + 1) ;
12 if CurrentBarIndex()=0 then
13   ema = close;
14 else
15   ema = factor * close + (1
```

- Um die Ausführung am Ende jeder Period anzuhalten, setzen Sie einen break()-Aufruf an das Ende der `Calculation` Klausel:

```
9  Calculation
10
11 factor = 2 / ($span + 1) ;
12 if CurrentBarIndex()=0 then //we need one lookback entry
13   ema = close;
14 else
15   ema = factor * close + (1 - factor) * ema[1];
16
17 Break();
18 interpretation TriggerLine(close, ema);
```

- plotten Sie eine Serie, die interne Zwischenberechnungen enthält
- nutzen Sie `Annotate()` oder `Highlight()`, um direkt in den Chart Werte zu schreiben
- nutzen Sie `ShowTip()`, um einer Periode ein Popup zuzuweisen:

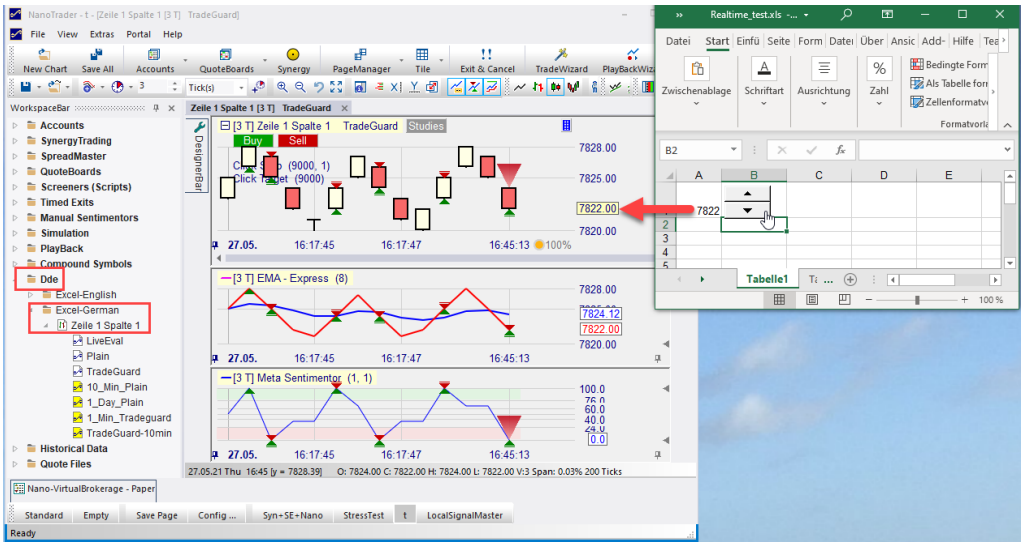


Denken Sie daran, dass Sie im Chart frei zoomen können, auch wenn ein Editor geöffnet ist. Dadurch lassen sich nach einmaligem Ausführen des Codes die Tips leicht inspizieren.

Um alle verwendeten Variablen anzuzeigen, rufen Sie `VarsToString()` mit einer leeren Zeichenkette auf:
`ShowTip(VarsToString(""));`

11.6 Verwenden von DDE zur Simulation bestimmter Datenkonstellationen

Oft müssen konkrete Datenkonstellationen getestet werden. Dazu ist es sehr komfortabel, die Daten via Excel selber zu generieren. Aktivieren Sie dazu unter Extras|Kursdatenversorgung den Eintrag DDE. Im Installationsverzeichnis finden Sie die Datei Realtime_test.xls. Öffnen Sie diese und fügen Sie den zu testenden Sentimentor in eine Studie ein, die auf Excel basiert.



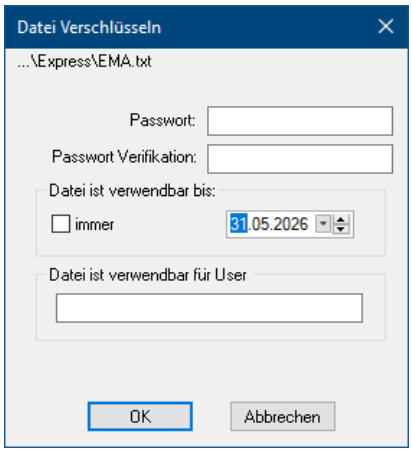
12 Express-Sentimentoren verschlüsseln

Express-Sentimentoren können auch verschlüsselt werden. Dies ermöglicht eine Vielzahl kommerzieller Nutzungsmöglichkeiten der Sentimentoren. Ein verschlüsselter Sentimentor kann ohne Einschränkung in Analysen verwendet werden – allerdings kann der Express-Code nicht eingesehen werden.

Um Express-Sentimentoren zu verschlüsseln, wählen Sie aus dem Menü Extras|Express-Sentimentoren verschlüsseln.

In dem erscheinenden Datei-Auswahl-Dialog können beliebig viele Dateien ausgewählt werden.

Anschliessend legen Sie das Passwort fest. Genau wie bei der Verschlüsselung von NanoTrader-Analysen kann optional ein Verfalltermin vorgegeben werden. Nach Ablauf des Verfalltermins kann der Sentimentor nicht mehr verwendet werden:



Die Einstellungen werden auf alle ausgewählten Sentimentoren angewendet.

Im Gegensatz zu unverschlüsselten Express-Sentimentoren werden die verschlüsselten Sentimentoren in Analysen *referenziert*, d.h. es wird auf die entsprechende Datei verwiesen. Bei Distribution einer Analyse, die verschlüsselte Sentimentoren verwendet, benötigt der Empfänger der Analyse also die verschlüsselter Sentimentor-Datei und muss diese in das Unterverzeichnis „Express“ des NanoTrader-Installationsverzeichnisses kopieren.

13 Verfügbare Funktionen und Prozeduren

Express stellt eine Bibliothek von Funktionen zur Verfügung, die innerhalb eines Express-Programms aufgerufen werden können. Falls eine aufgerufene Funktion Parameter benötigt, überprüft NanoTrader ob die übergebenden Parameter der *Funktionsdefinition* entsprechen. Ein *Funktionsdefinition* legt fest, wie die definierte Funktion aufzurufen ist.

Die Definition der Funktion `Max()` lautet beispielsweise:

```
float Max (float value1, float value2)
```

Der Rückgabewert der Funktion `Max()` ist vom Typ `float`. Die Funktion selbst erwartet zwei Parameter vom Typ `float`. Express konvertiert automatisch integer-Wert in float-Wert, so dass der Aufruf

```
Max (close, 5000)
```

ein gültiger Funktionsaufruf ist.

Funktionen, die einen Wert zurückgeben, können in Ausdrücken verwendet werden, z.B.

```
mySeries = Max (open, close) * 2;
```

Funktionen, die keinen Parameter zurückgeben, werden auch *Prozeduren* genannt. Eine Prozedur kann zwar nicht in Ausdrücken verwendet werden, bildet aber eine Anweisung:

```
MovingAverage (mySeries, mySeries, $span);
```

Die Definition von `MovingAverage` lautet

```
void MovingAverage (series source, series target, int span)
```

Der “Rückgabewert” `void` zeigt an, dass tatsächlich kein Wert zurückgegeben wird.

Die Namen der in den Funktionsdefinitionen verwendeten Variablen sind so gewählt, dass sie ihre Rolle für die Funktion umschreiben – ansonsten haben sie keine Bedeutung.

Auch wenn eine Funktion keine Parameter erwartet, müssen die Klammern im Funktionsaufruf angegeben werden:

```
index = CurrentBarIndex();
```

Die folgenden Funktionen stehen zur Verfügung:

Definition: float **AbsValue** (float wert)

Bedeutung: Liefert den absoulten Wert von `wert’.

Beispiel:
AbsWert (-3.7) liefert 3.7; AbsWert (5) liefert 5

Definition: void **Annotate** (string type, string color, float high, float low)
void **Annotate RGB** (string type, int red, int green, int blue, float high, float low)
void **AnnotateAt** (int offset, string type, string color, float high, float low)
void **AnnotateRGBAt** (int offset, string type, int red,int green,int blue, float high, float low)

Bedeutung: Fügt eine Anmerkung zur aktuellen Periode hinzu, wobei `type’ die zu verwendende Hervorhebungsart beschreibt und `color’ die zu verwendende Farbe.

Für `type’ können folgende Zeichenketten übergeben werden (siehe Bild):

“ellipse”, “upTriangle”, “downTriangle” sowie “labelLeft”, “labelCenter” und “labelRight”.

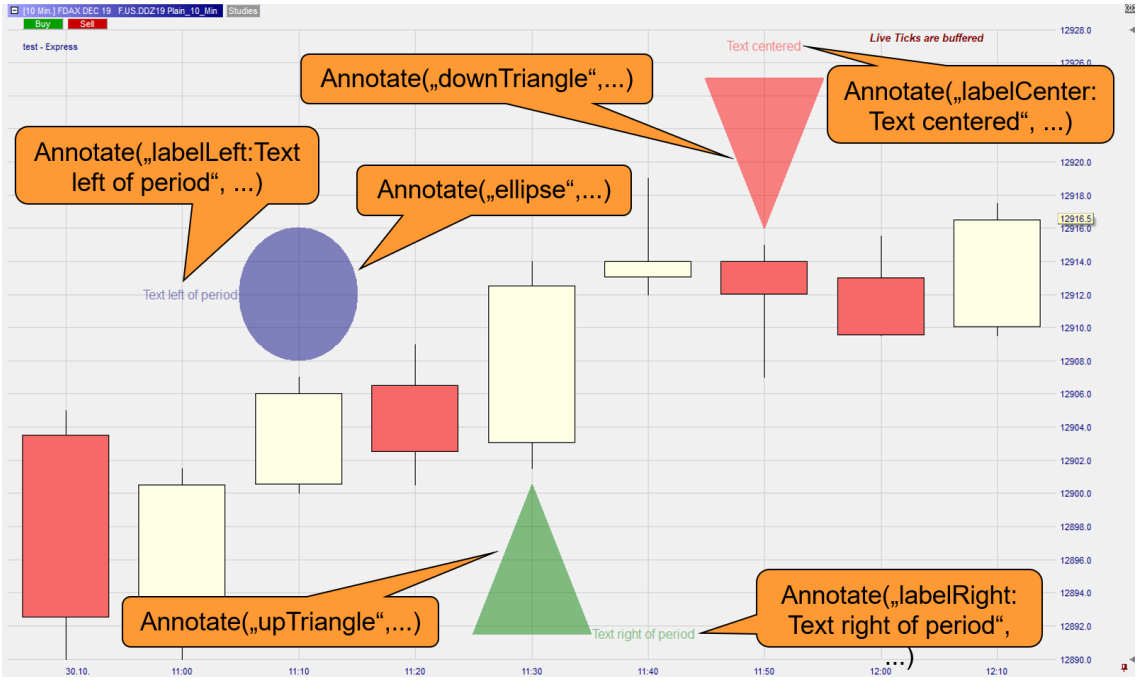
Die vertikale Position und Größe der Anmerkungen der Typen “ellipse”, “upTriangle” und “downTriangle” wird durch die Parameter `high’ und `low’ festgelegt.

Der Text, der mittels “labelLeft”, “labelCenter” und “labelRight” ausgegeben werden soll, wird durch einen Doppelpunkt getrennt mit übergeben. Beispiel:

```
Annotate(“labelLeft:This text appears\nleft of the period”, “black”, (open+close)/2, 0);
```

Ein Zeilenumbruch kann durch die Zeichenfolge `\n` erreicht werden. Die Zeilen werden nicht automatisch umgebrochen.

Die vertikale Position des Textes wird durch den Parameter `high' bestimmt.



Für eine Liste der unterstützten Farb-Bezeichnungen siehe Funktion „plot“.

Siehe Abschnitt "Mehr zum Thema Farben" um zu erfahren, wie Farben und ihre Deckkraft spezifiziert werden.

In einer Periode können mehrere Anmerkungen überlagert dargestellt werden.

Beispiel:

```
if (volume > 500) then
  Annotate("ellipse", "green,25", high, low); //opacity 25%
if (volume > 500) and IsBarCompleted() then
    Annotate("ellipse", "green", high, low); //no intra bar
                                             //annotation
```

Die „At“-Varianten ermöglichen das Hinzufügen einer Anmerkung relativ zur aktuellen Periode entsprechend dem Indizieren von Preis-Daten. Beispielsweise setzt AnnotateAt (2, „upTriangle“, „blue“, high, low) ein Aufwärtsdreieck zwei Perioden vor der aktuellen Periode. Dies erlaubt u.a. das Auszeichnen von Preismustern, die sich über mehrere Perioden erstrecken.

Siehe auch Highlight().

Definition: float **ArcTangent** (float wert)

Bedeutung: Liefert den Arcus Tangens von `wert'.

Beispiel: ArcTangent (2.7475) liefert 70.

Definition: float **Atr** (int span)

Bedeutung: Liefert die Arverage True Range des MasterCharts für die aktuelle und vorhergehenden `span' Perioden ausgedrückt in Prozent.

Definition: float **AtrAbs** (int span)

Bedeutung: Liefert die Average True Range des MasterCharts für die aktuelle und vorhergehenden `span' Perioden ausgedrückt in Punkten.

Definition: void **Bands** (series series, series lower, series upper)

Bedeutung: Standard Interpretations-Schema, wobei die Sentimente auf Basis des Kreuzen der oberen und unteren Bänder bestimmt wird.

Beispiel:

```
interpretation Bands (close, myLower, myUpper);
```

Definition: int **BarsSinceEntry** ()

Bedeutung: Vor wieviel Perioden wurde der aktuelle Trade eröffnet?

Beispiel:

```
if (MarketPosition() = 1) and (BarsSinceEntry() > 10) then
...
```

Definition: void **Break** ()

Bedeutung: Hält die Programmausführung an und ermöglicht die Überprüfung von Variablen. Die aktuelle Periode wird im Diagramm hervorgehoben.

Beispiel:

```
Break();
```

Diese Funktion ist nur aktiv, wenn der Express-Code über den Express-Editor ausgeführt wird. Ansonsten wird die Funktion ignoriert.

Definition: void **BreakIf** (bool value)

Bedeutung: Wie Break(), aber die Programmausführung wird nur angehalten, wenn `value' true ergibt.

Diese Funktion ist nur aktiv, wenn der Express-Code über den Express-Editor ausgeführt wird. Ansonsten wird die Funktion ignoriert.

Beispiel:

```
BreakIf ((CurrentBarIndex() > 50) and (open > close[1]));
```

Diese Funktion ist nur aktiv, wenn der Express-Code über den Express-Editor ausgeführt wird. Ansonsten wird die Funktion ignoriert.

Definition: void **BreakIfDrawingTool** ()

Bedeutung: Wie Break(), aber die Programmausführung wird nur angehalten, wenn in der gerade bearbeiteten Periode im MasterChart ein Zeichentool, beispielsweise ein Rechteck, beginnt oder endet.

Beispiel:

```
BreakIfDrawingTool();
```

Diese Funktion ist nur aktiv, wenn der Express-Code über den Express-Editor ausgeführt wird. Ansonsten wird die Funktion ignoriert.

Definition: void **CalculateAtEveryTick** (bool value)

Diese Funktion ist nur für Stop-Sentimentoren verfügbar.

Bedeutung: Durch Aufruf von CalculateAtEveryTick(false) wird das Express-Skript nicht bei jedem neuen Tick, sondern nur zum Ende einer Periode ausgeführt. Dies ermöglich eine enorme Performancesteigerung bei rechenintensiven oder unperformant implementierten Express-Sentimentoren.

Beispiel:

```
if IsFirstBar() then CalculateAtEveryTick(false);
```

Definition: float **Ceiling** (float value)

Bedeutung: Liefert die kleinste ganze Zahl, die größer als `value' ist.

Beispiel: Ceiling (2.95) liefert 3

Definition: float **Cosine** (float wert)

Bedeutung: Liefert den Kosinus von `wert' Grad.

Beispiel:

Cosine (45) liefert 0.7071

Definition: void **CreateFile**(string file, string text)

Bedeutung: Erzeugt eine Datei mit dem angegebenen Text als Inhalt. Wenn die Datei bereits existiert, wird der Text angefügt. Der Pfad und Dateiname werden über den Parameter `file' definiert.

Für eine präzise Beschreibung, wann exakt die Datei erzeugt wird siehe Funktion PlaySound(). Es gelten die gleichen Prinzipien.

Beispiel:

```
if (close > high[1]) and (close > high[2]) then
   CreateFile("C:\nano-actions.txt", TimeToString(datetime,
         "%Y-%m-%d %H:%M:%S") + " New peak at symbol " +
         SymbolName());
```

Definition: bool **CrossesAbove** (series curve, series trigger)

Bedeutung: Liefert true falls (curve[1] <= trigger [1]) and (curve > trigger), false sonst.

Beispiel:

```
if CrossesAbove (mySeries, close) then sentiment = 100;
```

Definition: bool **CrossesAboveThreshold** (series curve, float threshold)

Bedeutung: Liefert true falls (curve[1] <= threshold) and (curve > threshold), false sonst.

Beispiel:

```
if CrosseAboveThreshold (mySeries, 70) then
  sentiment = 100;
```

Definition: bool **CrossesBelow** (series curve, series trigger)

Bedeutung: Liefert true falls (curve[1] >= trigger[1]) and (curve < trigger), false sonst.

Beispiel:

```
if CrossesBelow (mySeries, close) then sentiment = 0;
```

Definition: bool **CrossesBelowThreshold** (series curve, float threshold)

Bedeutung: Liefert true (if curve[1] >= threshold]) and (curve < trigger), false sonst.

Beispiel:

```
if CrossesBelowThreshold (mySeries, 30) then sentiment = 0;
```

Definition: int **CurrentBarIndex** ()

Bedeutung: Der Index der aktuell bearbeiteten Periode. Die erste Periode hat den Index 0.

Beispiel:

```
highDiff = CurrentBarIndex() - IndexOfHighest (close, 10);
```

Definition: float **DateToNumeric** (time value)

Bedeutung: Konvertiert ein Datum in einen numerischen Wert. Das Datum 2021-04-23 wird in den Wert 130421 gewandelt.

Diese Funktion ist hilfreich, wenn Berechnungen in Abhängigkeit zum Datum durchgeführt werden sollen.

Beispiel:

```
sentiment = preCalculatedSentiment * (1 -
          (DateToNumeric(dateopen)-130000)/1231);
```

Definition: int **DayOfWeek** (time time)

Bedeutung: Der Index des Tages des von `time' bezeichneten Datums in der Woche. Montag = 1, Dienstag = 2, ...

Beispiel:

```
//keine Käufe am Freitag
if DayOfWeek(date) = 5 then sentiment = 50;
```

Definition: int **Duration** (time start, time end)

Bedeutung: Die Dauer zwischen `start' und `end' in Sekunden.

Beispiel:

```
d = Duration (timeOpen, time);
d = Duration (dateTime[20], dateTime);
```

Hinweis: Der Aufrufer muß sicherstellen, dass `start' und `end' von identischem Typ sind. Beispielsweise liefert Duration(dateOpen, time) ein falsches Resultat, da `dateOpen' nur die Datumskomponente enthält und `time' nur die Uhrzeit-Komponente.

Definition: float **EntryPrice** ()

Diese Funktion ist nur für Stop-Sentimentoren verfügbar.

Bedeutung: Der Einstiegskurs für die aktuelle Position. Falls der Stop im TradeGuard oder einer Taktik eingesetzt wird, wird der Preis bei *Aktivierung* des TradeGuards bzw. der Taktik geliefert. Falls der TradeGuard bereits aktiv ist, während die Position eröffnet wird, entspricht dies dem Füllpreis.

Siehe auch EntryPriceOriginal().

Beispiel:

```
SetStopPrice(EntryPrice() - 5 * TickSize());
```

Definition: float **EntryPriceOriginal** ()

Diese Funktion ist nur für Stop-Sentimentoren verfügbar.

Bedeutung: Der tatsächliche Füllpreis der aktuellen Position im Depot.

Siehe auch EntryPrice ().

Beispiel:

```
SetStopPrice(EntryPriceOriginal() + TickSize());
```

Definition: float **Exp** (float value)

Bedeutung: Die exp()-Funktion angewandt auf `value'.

Beispiel:

SetStopPrice(EntryPrice() – 0.05);

Definition: **ExpMovingAverage** (series source, series target, int span)

Bedeutung: Berechnet den `span'-Perioden exponentiellen gleitenden Durchschnitt von `source' und schreibt das Ergebnis nach `target'. Diese Funktion sollte *nur* in Verbindung mit IsFirstBar() oder IsFinalBar() verwendet werden.

Beispiel:

```
if IsFirstBar () then
  ExpMovingAverage (close, mySeries, $span);
if IsFinalBar () then
  ExpMovingAverage (mySeries, mySeries, $span);
```

Definition: int **FinalBarIndex** ()

Bedeutung: Liefert den Index der letzten auszuwertenden Periode.

Beispiel:

```
if IsFirstBar() then
begin
  ExpMovingAverage(close, ema1, 5);
  ExpMovingAverage(close, ema2, 33);
  for i = 0 to FinalBarIndex()
     helper[-i] = ema1[-i] – ema2[-i];
end
```

Definition: **ForceFlat** ()

Diese Funktion ist nur für Stop-Sentimentoren verfügbar.

Bedeutung: Bewirkt die Schließung der gesamten Position. Bei der Nutzung mehrerer Stops wird durch den Aufruf von ForceFlat() nicht nur der Teil der Position geschlossen, die durch den ForceFlat() aufrufenden Stop-Sentimentor geschützt wurde, sondern die gesamte Position.

Beispiel:

```
if (BarsSinceEntry() > 10)
    and (close < EntryPrice() + 50*TickSize()) then
  ForceFlat();
```

Definition: string **GetApplicationLanguage** ()

Bedeutung: Liefert die konfigurierte Sprache des NanoTrader. Mögliche Rückgabewerte sind DEU, ENG, FRA, HUN, ITA, NLD und POL.

Beispiel:

```
if GetApplicationLanguage() = "DEU" then
  MessageBox("Diese Nachricht ist auf Deutsch.");
```

Definition: int **GetArraySize** (array arr)

Bedeutung: Liefert die Anzahl Elemente des Arrays `arr'.

Beispiel:

```
sum = 0;
for i = 0 to GetArraySize(arr) - 1
begin
  sum = sum + arr[i];
end
```

Definition: int **GetDefaultOrderSize** ()

Bedeutung: Liefert die Ordergröße für das Instrument, auf dem das Express-Programm gerade angewendet wird. Diese Funktion ist nur im Experten-Modus verfügbar (siehe SetExpertsMode()).

Diese Funktion ist dann ideal, wenn die Ordergröße im Verhältnis zum Wert des Sentiments vergrößert/verkleinert werden soll.

Beispiel:

```
if (sentiment = 0) OR (sentiment = 100) then
   SetDefaultOrderSize(GetDefaultOrderSize()+1);
else
   SetDefaultOrderSize(1);
```

Definition: time **GetExpiration** ()

Bedeutung: Liefert den Verfallstermin für das Instrument, auf dem das Express-Programm gerade angewendet wird. Falls das Instrument keinen Verfallstermin hat, entspricht das Ergebnis der Funktion dem 1.1.1970 um 0:00 Uhr.

Beispiel:

```
if (date >= GetExpiration()) then
  sentiment = senti_flat;
```

Definition: string **GetPriceFormat** ()

Bedeutung: Liefert das interne Format für die Darstellung des Preises in der Y-Achse des MasterCharts.

Beispiel:

```
SetYscaleFormat (GetPriceFormat());
```

Nutze das gleiche Format im Indikator für die Darstellung des Preises, wie sie auch der MasterChart nutzt.

Definition: float **GetSpreadSize** ()

Bedeutung: Liefert den aktuellen Spread für das Instrument, auf dem das Express-Programm gerade angewendet wird.

Beispiel:

```
ask = close + GetSpreadSize();
```

Definition: float **Floor** (float value)

Bedeutung: Liefert die größte ganze Zahl, die kleiner als `value' ist.

Beispiel:

```
Floor (2.95) liefert 2
```

Definition: float **Highest** (series series, int span)

Bedeutung: Liefert den größten Wert der Elemente series[0], ..., series[span – 1].

Beispiel:

```
tenBarHigh = Highest (close, 10);
```

Definition: void **Highlight** (string type, string color)

void **HighlightRGB** (string type, int red, int green, int blue)

void **HighlightAt** (int offset, string type, string color)

void **HighlightRGBAt** (int offset, string type, int red,int green,int blue)

Bedeutung: Hebt die aktuelle Periode hervor, wobei `type' die zu verwendende Hervorhebungsart beschreibt und `color' die zu verwendende Farbe.

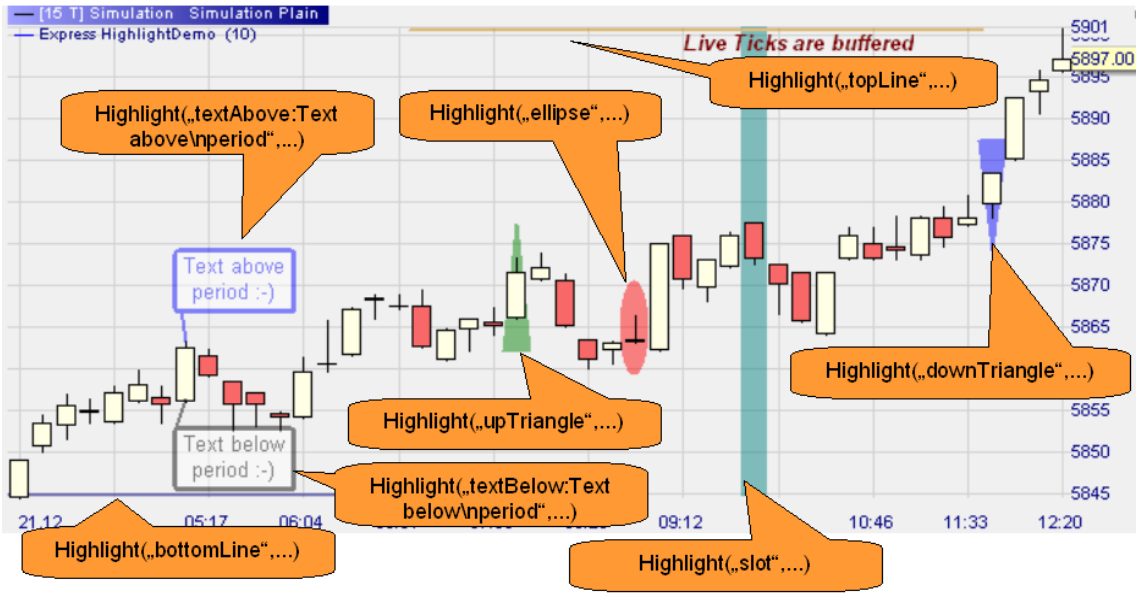
Für `type' können folgende Zeichenketten übergeben werden (siehe Bild):

“ellipse”, “upTriangle”, “downTriangle”, “slot”, “bottomLine”, “topLine”, sowie “textAbove”, “textBelow”.

Der Text, der mittels “textAbove” oder “textBelow” ausgegeben werden soll wird durch einen Doppelpunkt getrennt mit übergeben. Beispiel:

```
Highlight("textAbove:This text appears\nabove the period",
"black");
```

Ein Zeilenumbruch kann durch die Zeichenfolge `\n` erreicht werden. Die Zeilen werden nicht automatisch umgebrochen.



Für eine Liste der unterstützten Farb-Bezeichnungen siehe Funktion „plot“.

Siehe Abschnitt "Mehr zum Thema Farben" um zu erfahren, wie Farben und ihre Deckkraft spezifiziert werden.

In einer Periode können mehrere Highlights überlagert dargestellt werden.

Beispiel:

```
if (volume > 500) then Highlight("ellipse", "green");
//no intra bar highlighting:
if (volume > 500) and IsBarCompleted() then
    Highlight("ellipse", "green,25");  //opacity 25%
```

Die „At“-Varianten ermöglichen das Setzen eines Highlights relativ zur aktuellen Periode entsprechend dem Indizieren von Preis-Daten. Beispielsweise setzt HightlightAt(2, „upTriangle“, „blue“) ein Aufwärtsdreieck zwei Perioden vor der aktuellen Periode. Dies erlaubt u.a. das Markieren von Preismustern, die sich über mehrere Perioden erstrecken.

Definition: int **IndexOfHighest** (series series, int span)

Bedeutung: Liefert den absoluten Index des höchsten Wertes der Elemente series[0], ... series[span – 1]

Beispiel:

```
highDiff = CurrentBarIndex() - IndexOfHighest (close, 10);
```

Definition: int **IndexOfLowest** (series series, int span)

Bedeutung: : Liefert den absoluten Index des kleinsten Wertes der Elemente series[0], ... series[span – 1]

Beispiel:

```
lowDiff = CurrentBarIndex() - IndexOfLowest (close, 10);
```

Definition: bool **IsFinalBar**()

Bedeutung: Liefert true falls gerade die Berechnungen für die letzte Periode durchgeführt werden.

Beispiel:

```
if IsFinalBar () then
  MovingAverage (mySeries, mySeries, $span);
```

Definition: bool **IsFirstBar**()

Bedeutung: Liefert true falls gerade die Berechnungen für die erste Periode durchgeführt werden.

Beispiel:

```
if IsFirstBar () then
  MovingAverage (close, mySeries, $span);
```

Definition: bool **IsIntradayEntry**()

Diese Funktion ist nur für Stop-Sentimentoren verfügbar.

Bedeutung: Liefert true, falls die Position gerade in einer noch nicht abgeschlossenen Periode eröffnet wurde. In diesem Fall ist es oftmals nötig, eine andere Stoppreis-Berechnung durchzuführen, z.B. rein auf Basis des Einstiegspreises. Wird die Position nicht zum Periodenende glattgestellt, so wird nochmals ein Stopkurs berechnet, der dann für die nächste Periode verwendet wird.

Beispiel:

```
if IsIntradayEntry () then
  SetStopPrice(EntryPrice() - 0.05);
```

Definition: bool **IsNewDay**()

Bedeutung: Liefert true, falls die aktuelle Periode die erste Periode eines neuen Tages oder die erste Periode der betrachteten Datenreihe ist.

Beispiel:

```
if IsNewDay() then Highlight("slot", "blue");
```

Definition: bool **IsNonZero**(float value)

Bedeutung: Liefert true, falls `value' >= 0.001. Testen Sie in Berechnungen nie auf „= 0“, da aufgrund von Rundungsfehlern diese Bedingung fast nie eintritt.

Beispiel:

```
if IsNonZero(a * b) then val = sum / (a * b);
```

Definition: bool **IsZero**(float value)

Bedeutung: Liefert true, falls `value' < 0.001. Testen Sie in Berechnungen nie auf „= 0“, da aufgrund von Rundungsfehlern diese Bedingung fast nie eintritt.

Beispiel:

```
if Not IsZero(a * b) then val = sum / (a * b);
```

Definition: float **Log** (float wert)

Bedeutung: Liefert den natürlichen Logarithmus von `wert' oder void falls `wert' <= 0.

Beispiel:

```
Log (1000) liefert 6.9078
```

Definition: float **Lowest** (series series, int span)

Bedeutung: Liefert den kleinsten Wert der Elemente series[0], ... series[span – 1].

Beispiel:

```
tenBarLow = Lowest (close, 10);
```

Definition: int **MarketPosition** ()

Diese Funktion ist nur für Stop-Sentimentoren verfügbar.

Bedeutung: Liefert

1, wenn aktuell eine Long-Position vorlieg

0, wenn aktuell keine Position vorliegt

-1, wenn aktuell eine Short-Position vorliegt

Beispiel:

```
if MarketPosition() = 1 then SetStopPrice (low – 0.01);
```

Definition: int **MarketPositionSize** ()

Diese Funktion ist nur für Stop Sentimentoren verfügbar.

Bedeutung: Liefert das Volumen der aktuellen Position:

> 0 = long

0 = flat

< 0 = short

Stops, die MarketPositionSize() enthalten, werden für das Backtesting ignoriert.

Beispiel:

```
if MarketPositionSize() > 3 then
  SetStopPrice (low – 0.03);
```

```
else
  SetStopPrice (low – 0.01);
```

Definition: float **Max** (float wert1, float wert2)

Bedeutung: Liefert den größeren der Werte `wert1' und `wert2'

Beispiel:

Max (3, 7.5) liefert 7.5

Definition: float **MaxPriceEntryBar** ()

Diese Funktion ist nur für Stop-Sentimentoren verfügbar.

Bedeutung: Für Stops mit intraperioden-Updates (siehe SetIntraPeriodUpdate()). Liefert den höchsten Preis innerhalb der Eröffnungsperiode eines Trades, der *nach* der Positionseröffnung noch erreicht wurde.

Beispiel

```
if (MarketPosition() = 1) and IsIntradayEntry() then
  SetStopPrice (MaxPriceEntryBar() - 5 * TickSize();
```

Definition: float **MinPriceEntryBar** ()

*Diese Funktion ist nur für Stop-Sentimentoren verfügbar.*Bedeutung: Für Stops mit intraperioden-Updates (siehe SetIntraPeriodUpdate()). Liefert den kleinsten Preis innerhalb der Eröffnungsperiode eines Trades, der *nach* der Positionseröffnung noch erreicht wurde.

Beispiel:

```
if (MarketPosition() = -1) and IsIntradayEntry() then
  SetStopPrice (MinPriceEntryBar() + 5 * TickSize();
```

Definition: void **MessageBox**(string message)

Bedeutung: Stellt `message' in einem Popup-Fenster dar.

Für eine präzise Beschreibung, wann exakt die Meldung erscheint siehe Funktion PlaySound(). Es gelten die gleichen Prinzipien.

Beispiel:

```
if (close > high[1]) and (close > high[2]) then
  MessageBox("New peak at symbol " + SymbolName());
```

Definition: float **Min** (float wert1, float wert2)

Bedeutung: : Liefert den kleineren der Werte `wert1' und `wert2'

Beispiel:

Min (3, 7.5) liefert 3

Definition: **MovingAverage** (series source, series target, int span)

Bedeutung: Berechnet den `span'-Perioden gleitenden Durchschnitt von `source' und schreibt das Ergebnis nach `target'. Diese Funktion sollte *nur* in Verbindung mit IsFirstBar() oder IsFinalBar() verwendet werden.

Beispiel:

```
if IsFirstBar() then
  MovingAverage (close, mySeries, $span);
if IsFinalBar  () then
  MovingAverage (mySeries, mySeries, $span);
```

Definition: **NoDrawingOHLCinMC** ()

Bedeutung: Deaktiviert die Darstellung der Daten-Serien open, high, low und close (zum Beispiel in Bars, Renkos, Kerzen) im MasterChart. Diese Funktion ist nur im Experten-Modus verfügbar (siehe SetExpertsMode()).

Beispiel:

```
if IsFirstBar() then NoDrawingOHLCinMC();
```

Definition: **NormalCDF** (float value)

Bedeutung: Liefert den Wert der Dichtefunktion der Standardnormalverteilung an der Stelle `value'.

Beispiel:

```
cdf = NormalCDF(0.2);
```

Definition: **NormalPDF** (float value)

Bedeutung: Liefert den Wert der Standardnormalverteilung an der Stelle `value'.

Beispiel:

```
cdf = NormalPDF(0.2);
```

Definition: time **NumericToDate** (float value)

Bedeutung: Konvertiert `value' in einen time-Wert, wobei `value' als JJMMTT interpretiert wird. Zum Beispiel wird 130423 in den Zeitwert 2013-04-23 umgewandelt.

Falls der Tages-Teil von `value' größer als die Anzahl der Tage im angegebenen Monat ist, wird dieser auf den letzten Tag des Monats gesetzt. Falls der Monats-Teil von 'value' größer als 12 ist, wird er auf 12 gesetzt. Der Jahres-Teil von 'value' kann nicht größer als 99 sein.

Diese Funktion bietet sich an, wenn die auszuführende Berechnung auf einer Datumsangabe basiert, die über die DesignerLeiste einstellbar sein soll bzw. die in der Optimierung berücksichtigt werden soll.

Beispiel:

```
if (date >= NumericToDate($blockStart))
    and (date <= NumericToDate($blockEnd)) then
  sentiment = senti_block;
```

Definition: string **NumericToString** (float value, string format)

Bedeutung: Formatiert `value' gemäß der Formatangabe in `format'. `Format' unterstützt alle Format-Angaben der C-Funktion „printf()". Die wichtigsten Formate sind:

"%f"	Gleitkommadarstellung
"%6.2f"	rundet auf zwei Nachkommastellen
"%g"	kompakte Schreibweise
"%e"	wissenschaftliche Schreibweise

Falls `format' ein leerer String ist, wird das Format "%g" verwendet.

Für die Formatierung von Preisdaten siehe Funktion PriceToString().

Beispiel:

```
ShowTip(NumericToString(val, "%6.4f"));
```

Definition: time **NumericToTime** (float value)

Bedeutung: Konvertiert `value' in einen time-Wert, wobei `value' interpretiert wird als HHMM. Z.B. 1545 wird in den Zeitwert 15:45 gewandelt.

Falls der Stunden-Teil von `value' größer als 23 ist, wird er auf 23 gesetzt. Falls der Minuten-Teil von `value' größer als 59 ist, wird er auf 59 gesetzt.

Diese Funktion bietet sich an, wenn die auszuführende Berechnung auf einer Zeitangabe basiert, die über die DesignerLeiste einstellbar sein soll bzw. die in der Optimierung berücksichtigt werden soll.

Beispiel:

```
if (time >= NumericToTime($blockStart))
     and (time <= NumericToTime($blockEnd)) then
  sentiment = senti_block;
```

Definition: void **PlaySound**(string sound)

Bedeutung: Spielt die durch `sound' bezeichnete wav-Datei. `sound' kann ein kompletter Pfadname zu einer .wav-Datei sein oder lediglich den sogenannten „Datei-Titel" einer .wav-Datei bezeichnen, die sich im Unterverzeichnis Wav des Installationsverzeichnisses befindet. Enthält dieses Verzeichnis beispiesweise

eine Datei namens „ringin.wav“ würde ein Aufruf von PlaySound(„ringin“) diese Datei abspielen.

Ein Sound wird nur gespielt, wenn er durch eingehende Live-Daten aktiviert wird. Desweiteren wird er nur einmal gespielt.

Beachten Sie, dass das Express-Programm für jeden eingehenden Tick ausgeführt wird, d.h. innerhalb der aktuellen Periode wird der Sound gespielt sobald die Funktion PlaySound() aufgerufen wird – es wird mit dem Spielen des Sounds *nicht* bis zum Ende der Periode gewartet.

Falls ein Sound nur am Ende einer Periode gespielt werden soll, kann dies wie folgt erreicht werden:

```
if IsBarCompleted () and soundBedingung then
  PlaySound("gong");
```

Pro Periode wird höchstens ein Sound gespielt, nämlich der erste Sound der durch PlaySound() veranlasst wird.

Falls `sound’ nicht zu einer gültigen .wav-Datei aufgelöst werden kann, wird stattdessen ein „Beep“ gespielt.

Beispiel:

```
if volume > 300 then
  PlaySound("gong");  //intra bar-Benachrichtigung
                      //über ein hohes Volumen
if IsBarCompleted () and (close > high[1]) then
  PlaySound (“corkpop”);  //end of bar-Benachrichtigung
              //über einenPerodenschlusskurs, der das
              //vorige High übersteigt
```

Definition: void **Plot** (series curve, string color, int penWidth)
void **Plot** (series curve, int red, int green, int blue, int penWidth)

Bedeutung: Zeichnet die Serie `curve’ in der spezifizierten Farbe und Strichstärke.

Beispiel:

```
Plot (close, "green", 2);
Plot (close, 128, 128, 128, 1);    //grey
```

Definition: void **PlotBand** (series curve1, string color1, int penWidth1,
series curve2, string color2, int penWidth2, string fillColor

Bedeutung: Zeichnet Serien `curve1’ und `curve2’ und füllt den inneren Bereich mit der Farbe `fillColor’.

Beispiel:

```
PlotBand(upper, "green", 2, lower, "red", 2, "lightgreen");
```

```
PlotBand(upper, 150, 0, 0, 2, lower, 0, 0, 0, 2, 128, 128,
         128);
```

Definition: **PlotBars** (series open, series close, series high, series low)
PlotBars (series open, series close, series high, series low, string colorBull, string colorBear)
PlotBars (series open, series close, series high, series low, series colors)

Bedeutung: Zeichnet einen Bar-Chart unter Verwendung der übergebenen Serien. Sind keine Farben angegeben, werden die Einstellungen des FarbManagers für bullische und bärische Bars verwendet. Eine fast unendlich hohe Anzahl an Farben kann durch Nutzung der RGB-Notation gewählt werden.

Beispiel:

```
PlotBars (myOpen, myClose, high, low);
PlotBars (myOpen, myClose, high, low, "lightyellow",
          "lightblue");
PlotBars (myOpen, myClose, high, low, 150, 0, 0, 128, 128,
          128);
PlotBars (myOpen, myClose, high, low, myColors);
```

Definition: **PlotCandles** (series open, series close, series high, series low)
PlotCandles (series open, series close, series high, series low, string colorBull, string colorBear)
PlotCandles (series open, series close, series high, series low, series colors)

Bedeutung: Zeichnet einen Candle-Chart unter Verwendung der übergebenen Serien. Sind keine Farben angegeben, werden die Einstellungen des FarbManagers für bullische und bärische Candles verwendet. Eine fast unendlich hohe Anzahl an Farben kann durch Nutzung der RGB-Notation gewählt werden.

Beispiel:

```
PlotCandles (myOpen, myClose, high, low);
PlotCandles (myOpen, myClose, high, low, "lightyellow",
             "lightblue");
PlotCandles (myOpen, myClose, high, low, 150, 0, 0, 128,
             128, 128);
PlotCandles (myOpen, myClose, high, low, myColors);
```

Definition: void **PlotCrossingLines** (series curve1, string color1, int penWidth1, series curve2, string color2, int penWidth2, string fillColor1,

string fillColor2)

Bedeutung: Zeichnet Serien `curve1' und `curve2' und füllt den inneren Bereich mit der Farbe `fillColor1', wenn `curve1' höher als `curve2' ist. Andernfalls wird der innere Bereich mit der Farbe `fillColor2' gefüllt. Eine fast unendlich hohe Anzahl an Farben kann durch Nutzung der RGB-Notation gewählt werden.

Beispiel:

```
PlotCrossingLines(upper, "green", 2, lower, "red", 2,
                  "lightgreen", "lightred");
PlotCrossingLines(upper, 150, 0, 0, 2, lower, 0, 0, 0, 2,
                  128, 128, 128, 200, 200, 200);
```

Definition: void **PlotLine** (float wert, string color, int penWidth)

Bedeutung: Zeichnet eine horizontale Line auf Niveau `wert' in der spezifizierten Farbe und Strichstärke.

Beispiel:

```
PlotLine ($threshold, "red", 2);
PlotLine (100, 150, 0, 0, 1); //dark red
```

Definition: float **PointValue** ()

Bedeutung: Liefert den Wert eines Punktes des Symbols, auf das der Express-Sentimentor angewandt wird.

Beispiel:

```
barValue = (high - low) * PointValue();
```

Definition: float **Power** (float wert, float exponent)

Bedeutung: Liefert `wert' hoch `exponent'.

Beispiel:

Power (3, 3) liefert 27

Definition: float **PrevDayHigh/Low/Open/Close/Vol** ()

Bedeutung: Liefert den High/Low/Open/Close/Vol des Vortages bzw. `void` falls der Wert nicht verfügbar ist.

Beispiel:

```
yesterdayMedian = ( PrevDayHigh() + PrevDayLow() +
                    PrevDayClose() ) / 3
```

Definition: string **PriceToString** (float value)

Bedeutung: Rundet `value' zum nächstmöglichen Preis entsprechend der definierten TickSize und liefert das Ergebnis als String.

Berücksichtigt automatisch die Schreibweise in Bruchnotation.

Beispiel:

```
ShowTip("TriggerPrice = " + PriceToString(high[1] +
                        2*TickSize()));
```

Definition: int **RGB** (int red, int, green, int blue)

Bedeutung: Konvertiert die angegebenen Rot-, Grün- und Blauanteile einer Farbe in einen Integer-Zahlenwert, der die selbe Farbe spezifiziert. Diese Funktion ist hilfreich, wenn die einer Serie zugehörigen Farben als Parameter in PlotBars() und PlotCandles() übergeben werden sollen.

Beispiel:

```
myColors = RGB (255, 255, 160);
```

Definition: float **Round** (float value, int precision)

Bedeutung: Liefert`value' gerundet auf `precision' Nachkommastellen.

Beispiel:

Round (2.428, 2) liefert 2.43

Definition: float **RoundMultiple** (float value, float multiple)

Bedeutung: Liefert`value' gerundet auf das nächst gelegen Vielfache von `multiple'.

Beispiel:

```
RoundMultiple ((high + low) / 2, TickSize());
```

Definition: **RSI** (series source, series target, int span)

Bedeutung: Berechnet den `span'-Perioden RSI von `source' und schreibt das Ergebnis nach `target'. Diese Funktion sollte *nur* in Verbindung mit IsFirstBar() oder IsFinalBar() verwendet werden.

Beispiel:

```
if IsFirstBar () then RSI (close, mySeries, $span);
if IsFirstBar () then RSI (mySeries, mySeries, $span);
```

Definition: void **Screenshot**(string filename)

Bedeutung: Erstellt einen Screenshot des MasterCharts und speichert ihn im PNG-Format in ‘filename’ ab. Falls ‘filename’ keine vollständige Pfad-Angabe enthält, wird die Datei im Verzeichnis “Screenshots” abgespeichert.

Der Screenshot wird exakt so, wie auf dem Bildschirm dargestellt, erzeugt und hat die selbe Höhe sowie Breite.

Es wird empfohlen, die Funktion ScreenshotEx() zu verwenden, da diese über mehr Optionen verfügt.

Für eine präzise Beschreibung, wann exakt ein Screenshot erstellt wird, siehe Funktion PlaySound(). Es gelten die gleichen Prinzipien.

Beispiel:

```
If IsFinalBar() then
  Screenshot("epxress.png");
```

Definition: void **ScreenshotEx** (string filename, int style, int width, int height, int periodsOrDays)

Bedeutung: Erstellt einen Screenshot des MasterCharts bzw. Vermögensverlaufscharts und speichert ihn im PNG-Format in ‘filename’ ab. Falls ‘filename’ keine vollständige Pfad-Angabe enthält, wird die Datei im Verzeichnis “Screenshots” abgespeichert.

Der Parameter ‘style’ bestimmt die dargestellten Elemente und die verwendete vordefinierte Höhe sowie Breite des MasterCharts bzw. Vermögensverlaufscharts.

0 = exakt der Bildschirm-Darstellung entsprechend
1 = reduzierte Darstellung, mittlere Größe
2 = sehr reduzierte Darstellung, kleine Größe
3 = zeige Vermögensverlaufschart, mittlere Größe
4 = zeige Vermögensverlaufschart, kleine Größe

Benutzen Sie ‘width’ und ‘height’ um die Größe des erzeugten Screenshots anzugeben. Durch den Wert 0 wird die vom Parameter ‘style’ vorgegebene Größe verwendet.

‘periodsOrDays’ bestimmt die Anzahl der dargestellten Perioden, die von der neuesten Periode ausgehend gezählt werden. Ein Wert größer 0 definiert die Anzahl der dargestellten Perioden. Bei einem Wert von 0 wird der dargestellte Ausschnitt des MasterCharts bzw. Vermögensverlaufscharts verwendet. Bei Verwendung eines negativen Werts wird dieser als Anzahl der Tage interpretiert. Zum Beispiel bedeutet -2, dass der Screenshot die Daten von heute und gestern darstellen soll.

Für eine präzise Beschreibung, wann exakt ein Screenshot erstellt wird, siehe Funktion PlaySound(). Es gelten die gleichen Prinzipien.

Beispiel:

```
If IsFinalBar() then
  ScreenshotEx("epxress.png", 0, 600, 400, -2);
```

Definition: void **SendEmail**(string subject, string message)

Bedeutung: Sendet eine E-Mail mit `message' als Inhalt und `subject' als Betreff.

Für eine präzise Beschreibung, wann exakt die E-Mail gesendet wird siehe Funktion PlaySound(). Es gelten die gleichen Prinzipien.

Beispiel:

```
if (close > high[1]) and (close > high[2]) then
  SendEmail("NanoTrader-Notification",
            "New peak at symbol " + SymbolName());
```

Definition: void **SetArraySize** (array arr, int size)

Bedeutung: Setzt die Größe von Array `arr' auf `size' Elemente. Die Elemente können somit durch die Indizes 1 – (size –1) referenziert werden.

Beispiel:

```
if IsFirstbar() then SetSize(arr, $arrSize);
```

Definition: void **SetArrayTo** (array arr, float value)

Bedeutung: Setzt alle Elemente fon Array `arr' auf den Wert `value'.

Beispiel:

```
if IsFirstbar() then arr.SetArrayTo(arr, 500);
```

Definition: void **SetDefaultOrderSize** (int value)

Bedeutung: Setzt die Ordergröße auf `value' für das Instrument, auf dem das Express-Programm gerade angewendet wird. Da die Ordergröße pro Instrument eingestellt ist, wird die Änderung auf alle Konten und Studien angewendet, die dieses Instrument enthalten. Diese Funktion ist nur im Experten-Modus verfügbar (siehe SetExpertsMode()).

Diese Funktion ist dann ideal, wenn die Ordergröße auf dem Wert des Sentiments basieren soll.

Beispiel:

```
if (sentiment = 0) OR (sentiment = 100) then
   SetDefaultOrderSize(2);
else
   SetDefaultOrderSize(1);
```

Definition: void **SetExpertsMode** ()

Bedeutung: Rufen Sie SetExpertsMode() auf, um die Nutzung von Funktionen in Express zu aktivieren, die Experten-Kenntnisse erfordern (z.B. SetDefaultOrderSize()). Indem Sie den Experten-Modus aktivieren, bestätigen Sie, dass Sie wissen, was Sie tun. Die Verantwortung für jegliche Schäden, die durch Experten-Funktionen verursacht werden, liegt bei Ihnen.

Beispiel:

```
if IsFirstBar() then SetExpertsMode();
```

Definition: void **SetIntraPeriodUpdate** ()

Bedeutung: Nur für Stop-Sentimentoren anwendbar. Die Stop-Berechnung wird bei jedem neuen Tick ausgeführt. Dies ist besonders nützlich für die Programmierung von Taktiken.

Stops, die SetIntraPeriodUpdate() enthalten, werden für das Backtesting ignoriert.

Hinweis: Aus internen Gründen genügt das Vorhandensein der Funktion im Quelltext – auch wenn das Statement nicht ausgeführt wird. D.h. selbst bei If false then SetIntraPeriodUpdate(); wird die Intraperioden-Berechnung aktiviert!

Definition: void **SetLongTrigger** (float value)

Bedeutung: Setzt den Bestätigungskurs für eine Long-Position. Damit diese ausgewertet wird, ist im Evaluator die "Einstiegspolitik" auf "Bestätigungskurs nächste Periode" oder „Limit-Kurs nächste Periode“ zu setzen. Falls kein Express-Sentimentor diese Routine aufruft, wird das das Hoch/Tief der Signalperiode genommen. Falls mehrere Express-Sentimentoren Trigger-Kurse setzen, so wird der jeweils "strengste" Kurs gewählt.

Beispiel:

```
SetLongTrigger ((high + low) / 2);
```

Definition: void **SetShortTrigger** (float value)

Bedeutung: Analog SetLongTrigger().

Beispiel:

```
SetShortTrigger ((high + low) / 2);
```

Definition: void **SetStopPrice** (float value)

Bedeutung: Setzt den Stop für die kommende Periode bzw. für die aktuelle Periode, falls die Position gerade eröffnet wurde.

Hinweis: In einem Sentimentor darf nicht sowohl SetStopPrice() als auch SetTargetPrice() aufgerufen werden.

Beispiel:

```
SetStopPrice (low[-1]);
```

Definition: void **SetTargetPrice** (float value)

Bedeutung: Setzt das Gewinnziel für die kommende Periode bzw. für die aktuelle Periode, falls die Position gerade eröffnet wurde.

Hinweis: In einem Sentimentor darf nicht sowohl SetStopPrice() als auch SetTargetPrice() aufgerufen werden.

Beispiel:

```
SetTargetPrice (high [-1]);
```

Definition: void **SetYscaleFormat** (string format)

Bedeutung: Setzt das Format für die Y-Achse gemäß printf..

Beispiel:

```
SetYscaleFormat("%.5d"); //nutze 5 Nachkommastellen.
SetYscaleFormat("%g"); //verwende die jeweils kompakteste
                       //Schreibweise.
```

Definition: void **ShowTip** (string message)

Bedeutung: Ordnet den String `message' der aktuellen Periode zu. Die Nachricht wird einem Popup-Fenster dargestellt, wenn der Cursor im MasterChart oder dem zugehörigen Chart-Fenster auf die betreffende Periode zeigt. Um einen Zeilenumbruch in der Nachricht zu erwirken muss die Zeichenfolge `\n` verwendet werden.

Beispiel:

```
if CrossesAbove (mySeries, 70) then
  ShowTip ("Eintritt in obere Zone!\nWarte auf
Bestätigung.");
```

Definition: float **Sign** (float wert)

Bedeutung: Liefert das "Vorzeichen" von `wert'.

Beispiel:

Sign (-3) liefert –1; Sign (5) liefert 1; Sign (0) liefert 0.

Definition: float **Sine** (float wert)

Bedeutung: Liefert den Sinus von `wert' Grad.

Beispiel:

Sine (70) liefert 0.9397

Definition: float **SquareRoot** (float wert)

Bedeutung: Liefert die Quadratwurzel von oder `void` falls `wert' < 0.

Beispiel:

SquareRoot (4) liefert 2

Definition: float **StdDev** (series source, series target, int span)

Bedeutung: Liefert die Standardabweichung der Werte source, source [1], ..., source[span-1] und schreibt das Ergebnis in target.

Diese Funktion sollte *nur* in Verbindung mit IsFirstBar() verwendet werden.

Beispiel:

```
StdDev (close, myseries, 10);
```

Definition: float **Sum** (series series, int span)

Bedeutung: : Liefert die Summe der Elemente series[0], ... series[span – 1]. Liefert `void` falls eines oder mehrere der Element `void` sind.

Beispiel:

```
amount = Sum (mySeries, $span);
```

Definition: void **Swing** (series series, input spanLeft, input spanRight)

Bedeutung: Standard Interpretations-Schema das die Sentimente auf Basis von Aufwärts-und Abwärtsschwüngen von `series' bestimmt.

Beispiel:

```
interpretation Swings (mySeries);
```

Definition: string **SymbolName**()

Bedeutung: Liefert den Namen des Symbols, auf den das Express-Programm gerade angewandt wird.

Beispiel:

```
if (close > high[1]) and (close > high[2]) then
  MessageBox("New peak at symbol " + SymbolName());
```

Definition: float **Tangent** (float value)

Bedeutung: Liefert den Tangens von ` value Grad.

Beispiel:

Tangent (70) liefert 2.7475.

Definition: float **TickSize** ()

Bedeutung: Liefert die Größe eines Ticks des Symbols, auf das der Express-Sentimentor angewandt wird.

Beispiel:

```
trigger = high[1] + 3 * TickSize();
```

Definition: float **TickValue** ()

Bedeutung: Liefert den Wert eines Ticks des Symbols, auf das der Express-Sentimentor angewandt wird.

Beispiel:

```
profit = $nbTicks * TickValue();
```

Definition: float **TimeToNumeric** (time value)

Bedeutung: Konvertiert eine Uhrzeit in einen numerischen Wert. Die Uhrzeit 15:45 wird in den Wert 1545 umgewandelt.

Diese Funktion ist hilfreich, wenn Berechnungen in Abhängigkeit zur Uhrzeit durchgeführt werden sollen.

Beispiel:

```
sentiment = preCalculatedSentiment *
             ( 1 - TimeToNumeric(timeopen) /2400);
```

Definition: string **TimeToString** (time timeVal, string format)

Bedeutung: Formatiert `timeVal' gemäß der Formatangabe in `format'. `format' unterstützt alle Formatierungen der C-Funktion `strftime(), d.h.:

%a abgekürzter Wochentagname
%A voller Wochentagname
%b abgekürzter Monatsname
%B voller Monatsname
%c Datum/Zeit-Darstellung gemäß Landeseinstellung
%d Tag im Monat als Nummer (01 – 31)
%H Stunde im 24-Stunden Format (00 – 23)
%I Stunde im 12-Stunden Format (01 – 12)
%j Tag im Jahr als Nummer (001 – 366)
%m Monat als Nummer (01 – 12)
%M Minute als Nummer (00 – 59)
%p A.M./P.M. für 12-Stunden Format
%S Sekunde als Nummer (00 – 59)
%U Woche im Jahr als Nummer wobei Sonntag der erste Wochentag ist (00 – 53)
%w Wochentag als Nummer (0 – 6; Sunday is 0)
%W Woche im Jahr als Nummer wobei Montag der erste Wochentag ist (00 – 53)
%x Datum gemäß Landeseinstellung

%X Zeit gemäß Landeseinstellung
%y Jahr ohne Jahrhunderte (00 – 99)
%Y volle Jahresbezeichnung
%z, **%Z** Name der Zeitzone
%% Prozent-Zeichen

Falls `format' leer ist, wird automatisch Datum und Zeit angezeigt.

Beispiel:

```
ShowTip(TimeToString(time, "%H:%M:%S"));
```

Definition: **TriggerLine** (series curve, series trigger)

Bedeutung Standard Interpretations-Schema das die Sentimente auf Basis von Kreuzungen der Serien `curve' und `trigger' bestimmt.

Beispiel:

```
interpretation TriggerLine (close, mySeries);
```

Definition: void **TwoThresholds** (series series, input upThreshod, input down-Threshold)

Bedeutung: Standard Interpretations-Schema, das die Sentimente auf Basis des Ein-und Austretens von `series' in eine obere bzw. untere Zone bestimmt.

Beispiel:

```
interpretation TwoThresholds (mySeries, $upperZone,
$lowerZone);
```

Definition: void **VarsToString** (string vars)

Bedeutung: Liefert die aktuellen Werte der Variablen, die in `vars‘ als kommagetrennte Liste angegeben sind.

Wenn 'vars' leer ist, dann werden die Werte *aller verwendeten* Variablen zurückgegeben, wobei "verwendet" bedeutet, dass die Variablen nicht nur definiert sind, sondern tatsächlich auch referenziert werden.

Das $-Zeichen, das zur Kennzeichnung von Eingabeparametern verwendet wird, ist optional.

Bei einer Serie wird der Wert der aktuellen Periode zurückgegeben.

Array-Variablen werden ignoriert.

Beispiele:

```
ShowTip(VarsToString("close,ma,$input"));
ShowTip(VarsToString("close,ma,input"));
ShowTip(VarsToString(""));
```

Definition: void **Unaggregate** (series source, series target)

Bedeutung: Falls die Serie `source' von einem anderen Sentimentor importiert wird, der wiederum aggregiert ist, kann Unaggregate() dazu verwendet werden, die Serie zurück auf die MasterChart-Aggregation abzubilden.

Beispiel:

```
series maAgg(MovingAverage.main);  //der MovingAverage
          //Sentimentor ist in der Studie aggregiert
series ma;
...
if IsFirstBar() then
  Unaggregate (maAgg, ma);
```

Definition: **WeightedMovingAverage** (series source, series target, int span)

Bedeutung: Berechnet den `span'-Perioden gewichteten gleitenden Durchschnitt von `source' und schreibt das Ergebnis nach `target'. Diese Funktion sollte *nur* in Verbindung mit IsFirstBar() oder IsFinalBar() verwendet werden.

Beispiel:

```
if IsFirstBar () then
  WeightedMovingAverage (close, mySeries, $span);
if IsFinalBar () then
  WeightedMovingAverage (mySeries, mySeries, $span);
```