NanoTrader Express

Language Reference

Document Version 3.4.1

www.fipertec.com

TABLE des MATIERES

1 Introduction

Nano Trader-Express est un module qui permet de programmer des sentimentors, des stops et des filtres. Ceux-ci peuvent être utilisés exactement de la même façon que les sentimentors existant en base dans le programme, par exemple, ils peuvent être combinés avec d'autres sentimentors et, bien sûr, optimisés.
Nano Trader et son environnement Express offre des possibilités inégalées pour spécifier, optimiser, contrôler en BackTesting, et finalement appliquer vos idées de trading.

Notez qu'il n'est pas nécessaire d'avoir une licence d'exploitation spécifique pour bénéficier des avantages d'Express. Vous pouvez utiliser Express pour calculer et tracer dans un graphique des indicateurs, et également créer des annotations graphiques ou émettre des messages et des alarmes.

Le but de ce document est de décrire le langage de NanoTrader-Express, et non de présenter de façon générale les règles et la pratique de la programmation ou de constitution des algorithmes. Le lecteur non familier avec ces règles se reportera utilement à la littérature existante, traitant, par exemple, de Visual Basic, Pascal, Programmation en Excel, ou "Easy Language for Trade Station. Une fois les principaux concepts assimilés, tels que les V*ariables,* les B*oucles*, les E*xpressions Conditionnelles*, il sera facile de travailler avec Express.

Une connaissance de la terminologie utilisée dans NanoTrader, présentée dans le manuel *Graphisme et Trading***,** est également nécessaire.

Attention, un sentimentor programmé dans Express, ne conduit pas directement à une action de trading. En effet, Nano Trader génère les signaux en fonction d'une approche globale utilisant la valeur de sentiment du "Meta Sentimentor". Celui-ci agrège, en les pondérant éventuellement, les différents sentimentors utilisés dans l'étude. Un sentimentor programmé dans Express est alors un sentimentor qui, au même titre que les autres, est pris en compte dans le MetaSentimentor. C'est la valeur de ce dernier qui, en fonction de l'approche de trading retenue, va, ou non, déclencher une action de trading
.

2 Express – Fonction "Exécuter seulement"

Par contre, accéder à la programmation de Express requiert une licence spécifique. Cependant, même sans cette licence, vous pouvez exécuter ou voir les scripts de Express. Mais vous ne pourrez ni les modifier ni en créer de nouveaux. C'est ce qu'on nomme la fonction "exécuter seulement"
Pour qu'un utilisateur sans licence puisse utiliser, et seulement utiliser, un Script de Express (ou une étude incluant un script Express), ce script doit être ouvert au moins une fois dans l'éditeur Express par un utilisateur possédant la licence. Le script se

voit alors adjoindre une marque cachée dont la présence permettra ensuite que le script soit exécuté par des "utilisateurs sans licence"

Nous encourageons les programmeurs à mettre leurs scripts à disposition de la communauté des utilisateurs de Nano Trader.

3 Structure d'un Programme Express

Un Express sentimentor se calcule en suivant les étapes ci-après:

- Déclarer et initialiser les variables nécessaires au calcul
- Pour chaque barre (bougie) effectuer les calculs requis
- Calculer les valeurs du "sentiment", c'est-à-dire les règles d'interprétation du calcul précédent
- Définir le (ou les) graphique(s) devant être tracé(s).

Un programme Express va comporter les sections suivantes:

`Variable Declarations Section:`	Section Déclaration des variables
`Calculation Section:`	Section Calculs
`Interpretation Section:`	Section Interprétation
`Plot Section :`	Section Affichage

4 Exemple d'un Sentimentor

Prenons un exemple simple : celui du sentimentor basé sur une Moyenne Exponentielle (EMA, Exponential Mobile Average). Les signaux, d'Achat ou de Vente, sont déclenchés lorsque la courbe EMA croise la courbe des prix de clôture. Ce sentimentor fait partie de la liste des sentimentors inclus en base dans Nano Trader.

(Indication : ce code ne sert que d'exemple afin de monter la syntaxe exact utilisée par Express)

```
//(c) Fipertec                                                    1.

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

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

if close[1] = void then  // regarder une bougie en amont          9.
  ema = close;                                                    10.
```

```
else
  ema = factor * close + (1 - factor) * ema[1];          11.

interpretation TriggerLine(close, ema);                  12.

plot (ema, "blue", 2);                                   13.
plot (close, "black", 1);                                14.
```

Explication :

1. *Un double slash* "//" est placé en tête de ligne pour introduire un commentaire ne dépassant pas la fin de ligne. On peut aussi utiliser les parenthèses accolades { } pour insérer un commentaire n'importe où dans le code, ou pour désactiver une partie du code.
2. Le programme démarre toujours avec *Express <name>,* expression dans laquelle *Name* est le nom attribué au sentimentor (ici : EMA), nom que l'on retrouvera dans la Barre de Personnalisation.
3. Le mot clef *Vars* marque le début de la zone de Déclaration des Variables (*Variable Declaration Section)*
4. Cette ligne décrit une variable exprimée par un nombre entier *(*integer*);* le caractère $ indique qu'il s'agit d'une variable sujette à optimisation, *span* est le nom de la variable tel qu'il apparaitra dans la Barre de Personnalisation, les chiffres 1 et 200 sont les valeurs minimum et maximum que peut prendre la variable, 10 est la valeur initiale.
A présent le mot 'span' s'affiche automatiquement dans la Barre de Personnalisation en tant que variable et son nom peut être changé.
5. *Factor* est défini en tant que valeur numérique pouvant prendre des valeurs entières ou décimales. Le chiffre (0) initialise la valeur à zéro. Cependant lorsque Express initialise une valeur à zéro, le (0) peut être omis. La case des caractères, majuscules ou minuscules, n'a pas d'importance dans Express, c'est à dire que *factor, Factor, FACTOR* renvoient tous à la même variable. Il en est de même pour les mots clés.
6: *ema* est défini en tant que *series* de données décimales. Une série a la même longueur que celle analysée dans le graphique. Les éléments des séries sont automatiquement initialisés à zéro. Si on utilise le mécanisme *(val)*, *val* représente la valeur initiale de tous les éléments de la série.
7: le mot *calculation* est un mot réservé. Il introduit les règles de calcul qui vont être appliquées. Les calculs sont faits successivement pour chaque barre, en commençant par la plus ancienne ou la plus à gauche.
8 : la règle de calcul de *factor* est indiquée. Le calcul est basé sur la valeur de la variable d'entrée *$span*. Pour les opérations, on utilise les opérateurs usuels (+, --, *, /) ainsi que les parenthèses ( ).
9 : certaines séries sont déjà définies dans Express - par exemple *Close, Open, High, Low, Volume* – donnant accès aux données des graphiques. Le calcul de la valeur de EMA pour la période en cours nécessite de connaître la valeur de la période précédente (voir formule de EMA ligne 11). Pour désigner les périodes précédentes on utilise la notation suivante : *close [1]* est la clôture de la période immédiatement précédente; plus généralement, *close[n]* est la valeur de clôture de la période située n périodes en amont, *close* est synonyme de *close[0]*.

Supposons maintenant que l'on veuille calculer l'EMA de la toute première période. Ce n'est pas possible puisque *close[1]* n'a pas de signification. La valeur de *close[1]* sera alors notée *void* , mot réservé au cas où une donnée n'existe pas.(Surtout ne pas confondre *void* et valeur "0")
On va utiliser l'expression conditionnelle *If ….Then* (Si…Alors). La partie *Then* n'est exécutée par le programme que si la condition *If* est vérifiée.
Dans le cas où la partie *Then* comprend plus d'un cas possible, la partie *Then* doit commencer par *begin* et se terminer par *end* (voir exemple §7.11)

10 : la valeur initiale de l'EMA est spécifiée pour la première barre (ou bougie). Dans le cas de l'exemple on a, ligne 9 : *If close[1]= void then*, et ligne 10 *ema=close*. Ce qui veut dire que pour la première bougie du graphique on considère que EMA= valeur de clôture de la bougie.(On a défini la condition ligne 9, et on dit ligne 10 ce qu'il faut faire si la condition est remplie)

11: formule de calcul de Ema pour la période courante.
La formule est précédée de *else* ("autrement") car c'est le calcul qui doit être fait lorsque l'on ne se trouve pas dans le cas décrit par la condition précédente *If*.

12: *interpretation TriggerLine* correspond à la conversion en un sentiment des croisements EMA /cours de clôture, selon la règle prédéfinie dans les schémas d'interprétation en cas de croisement des cours (*close*) avec une ligne de déclenchement (*ema*) (Trigger Line). (On pourra se reporter au chapitre 5 du manuel Systèmes de Trading).
Autre possibilité : calculer explicitement les sentiments (= la valeur du sentimentor) en affectant les valeurs de sentiments des "séries *sentiment*" prédéfinies.

Dans le cas où un schéma d'interprétation n'est pas utilisé, il est possible de créer une valeur de sentiment avec un code individuel dans la section interprétation. Dans ce cas les valeurs des sentiments doivent être attribuées à la série prédéfinie « sentiment » et le code doit toujours commencer avec « begin » et se terminer par « end »

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

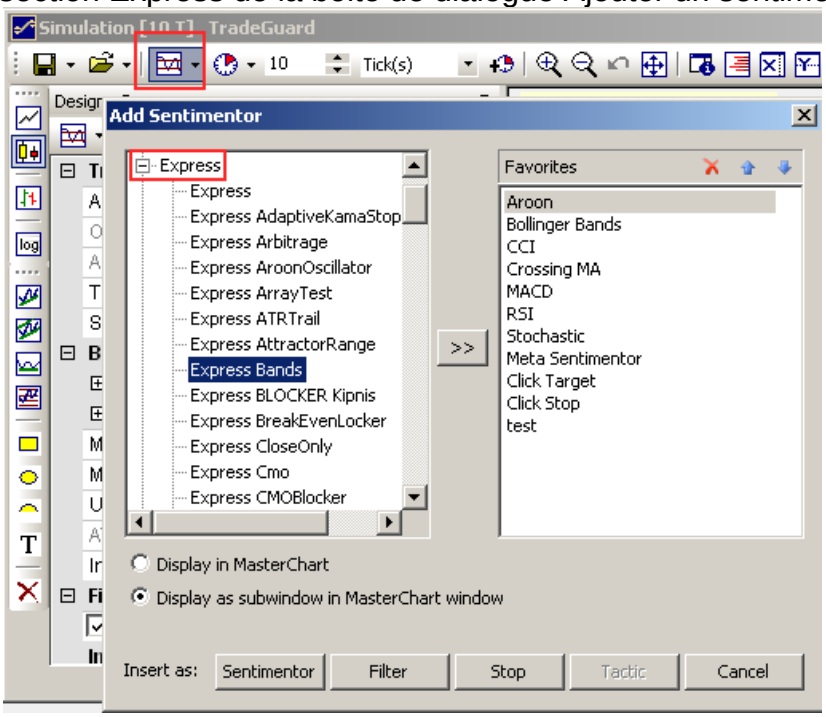
13: la série *ema* doit être dessinée en bleu, épaisseur de trait = 2

14 : la série *close* doit être dessinée en noir, avec une épaisseur de trait = 1

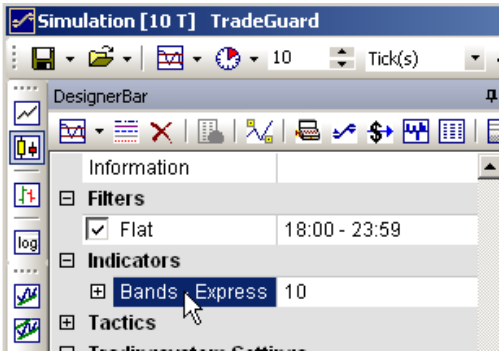
Note: afin de respecter la syntaxe de programmation, chacune des lignes de déclaration d'une variable, d'une formule de calcul ou d'une condition (If…Then …; Else …), d'une interprétation, ou d'un dessin (plot..) *doit se terminer par un point virgule*.

5 Ajouter un Express-Sentimentor à une étude

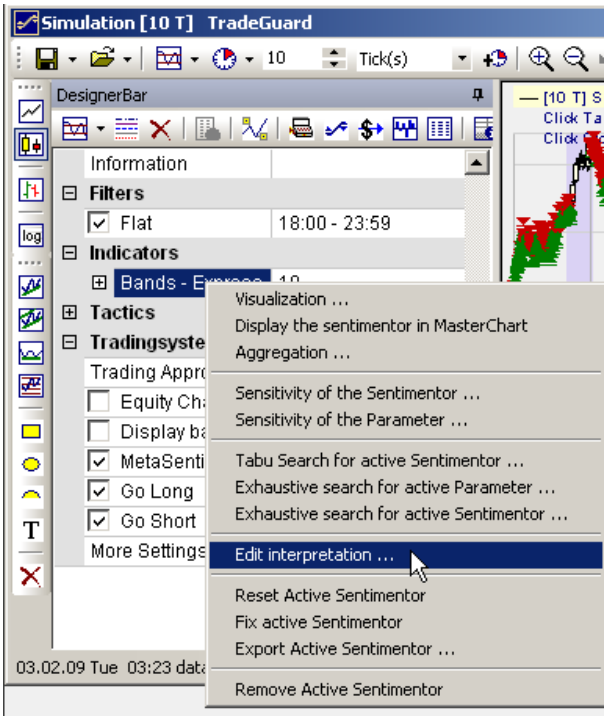
Pour ajouter un Express-Sentimentor à une étude, sélectionnez le sentimentor concerné dans la section Express de la boite de dialogue Ajouter un sentimentor :



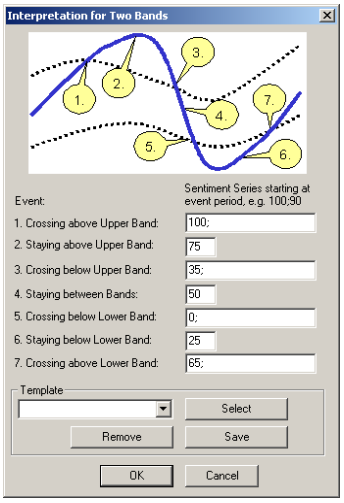
Pour éditer le code d'un Express sentimentor, double cliquez dans la ligne correspondante de la Barre de Personnalisation.



Dans le cas où l'Express-Sentimentor utilise par défaut un schéma d'interprétation (voir ci-dessous), le schéma peut être configuré en cliquant sur l'icône (ou en cliquant droit sur le nom de l'Express-Sentimentor dans la Barre de Personnalisation), puis choisir Modifier Interprétation dans le menu contextuel :



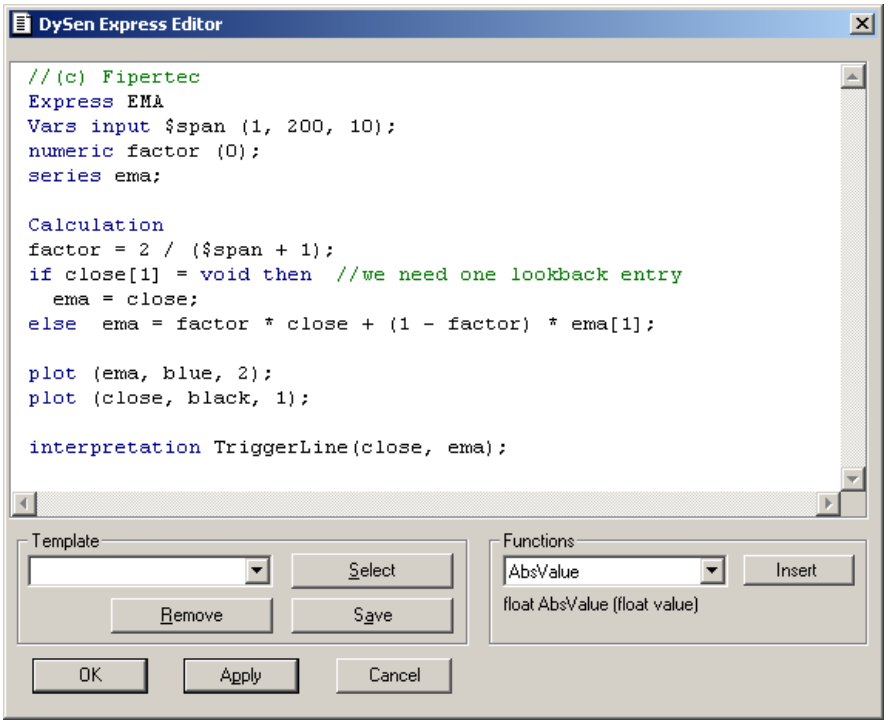
Ceci affiche l'Editeur de Sentiments :



Vous trouverez une description plus détaillée du schéma d'interprétation dans le document 'Trading Systems'.

6 Utiliser l'Express Editeur

Un double clic sur le nom d'un Express Sentimentor dans la Barre de Personnalisation, ouvre l'Express éditeur.



Les mots réservés sont affichés en bleu, et les commentaires en vert.

6.1 Raccourcis Clavier

Les raccourcis clavier de Windows pour Couper ou Coller du texte s'appliquent, par ex : Ctrl+C pour copier, Ctrl+V pour coller, Ctrl-A pour sélectionner tout le texte, Ctrl-S pour sauvegarder le texte.
On peut aussi utiliser de façon illimitée les raccourcis Ctrl+z pour Annuler, et Ctrl+Y pour Répéter.

6.2 Vérifier un Programme Express

Cliquer sur le bouton "Appliquer" lance le programme Express; les résultats en termes de calculs de séries, de tracés, de signaux sont immédiatement affichés dans les graphiques.
Si le code contient des erreurs, Nano Trader affiche un message d'erreur approprié.

6.3 Comment Enregistrer un programme Express

En cliquant sur le bouton OK, vous quittez l'éditeur Express. Ce programme est associé à l'Express Sentimentor sur lequel vous avez double-cliqué pour démarrer l'éditeur. Vous pouvez sauvegarder l'étude en cliquant sur le bouton de la fenêtre de l'écran graphique. A défaut le programme sera perdu.

La méthode habituelle consiste à sauvegarder le programme en tant que modèle avant de quitter l'éditeur Express. Le programme est placé dans le répertoire

"Express" en dessous du répertoire d'installation de NanoTrader, et devient disponible en tant que sentimentor dans la boite de dialogue "Ajout d'un sentimentor" et dans la fenêtre de l'éditeur.

Vous pouvez, depuis l'éditeur, charger tout modèle et l'adapter à vos besoins propres.

Il est recommandé, lors de l'enregistrement en tant que modèle, d'utiliser, même si le système ne l'impose pas, le nom choisi pour le programme Express. Supposons que le programme démarre avec la mention *Express UltimateSenti* C'est ce nom, *UltimateSenti,* qui deviendra le nom du sentimentor et apparaitra dans la Barre de Personnalisation.

7 Particularités du Langage Express

7.1 Types

Express est ce qu'on appelle un langage à types, c'est-à-dire que chaque entité représentant une valeur correspond à un certain type, ou catégorie, ce qui permet d'éviter immédiatement des erreurs de programmation.

Express utilise les types suivants:

- *numeric*
  Une valeur "*numéric*" s'exprime par un nombre entier (ex: 10) ou décimal (ex3.75).
  Si nécessaire NanoTrader convertit les valeurs décimales en nombres entiers en supprimant la partie décimale.

- *series*
  Une *serie* est une suite d'éléments de type *numeric*. Si, par exemple, on analyse un graphique comportant 200 bougies, alors une *serie* aura aussi 200 éléments. Si le graphique principal se constitue progressivement au fur et à mesure qu'arrivent des données en tant réel, il en sera de même pour les *series*.

- *array*
  Un array correspond à tableau comportant un nombre déterminé d'éléments de type numeric. La taille d'un array ne change pas automatiquement lorsque la série se constitue.
  Les valeurs d'entrée d'un array sont initialisées à 0.

- *string*
  Un *string* est une séquence de caractères, comme, par exemple, *"Hello World !"*. Lorsqu'on utilise un string en Express, les caractères doivent être insérés entre des "guillemets".

- *bool*
  Une entité booléene peut prendre 2 valeurs: *True* (Vrai) ou *False* (Faux).

- *time*
  L'entité *time* (Temps) recouvre à la fois des heures ou des dates.Ex :
  if time open <10:00then sentiment = 50;
  if date =2_8_2002 then sentiment = 100;

7.2 Mots réservés

Express utilise un certain nombre de mots pour lesquels la signification est bien précise. Ces mots ne peuvent pas être utilisés pour nommer des variables.

Les mots réservés sont :
express, sentimentor, blocker, stop, vars, calculation,interpretation, numeric, bool, info, export, 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.

Les mots suivants sont réservés pour un usage ultérieur :
function, procedure, import, plothistogram

7.3 Les Expressions

Une expression est une combinaison d'opérateurs et d'opérants.
Exemple: dans l'expression 5 + 3, les nombres 5 et 3 sont des opérants, le signe + est un opérateur.
Plus généralement, la valeur d'une variable, une série, ou la valeur résultat d'une fonction sont aussi des expressions.
Express distingue 3 types d'expressions appelées numériques (numeric), string (string), booléenne (boolean).

7.3.1 Les expressions numériques

Les expressions numériques combinent les opérateurs arithmétiques, +, -, * , / avec les règles habituelles; exemple : 5*close – 3*close[1]

On utilise des parenthèses pour grouper des expressions;
exemple: (high + low + close) / 3

7.3.2 Expressions relationnelles

Une expression relationnelle va vérifier si la proposition énoncée est True ou False , c'est-à-dire *vraie* ou *fausse*, comme dans l'expression close>open.

Les opérateurs relationnels sont :

Opérateur	Signification
>	plus grand que
<	plus petit que
=	égal à
> =	supérieur ou égal à
< =	inférieur ou égal à
< >	différent de

Les expressions relationnelles s'utilisent avec des entités numériques, de temps (de date ou d'heure) ou des expressions string. Dans ce dernier cas c'est l'ordre alphabétique qui détermine la relation opérationnelle; on dira par exemple abc < xyz, ou ada > acx.

7.3.3 Expressions logiques

Une expression logique évalue si une proposition est True ou False, c'est-à-dire Vraie ou Fausse, en combinaison avec les opérateurs and et or (et … ou). Exemples:
(close > open) and (volume <= 1000)
(close > close[1] and ((open > close) or (volume > 1000))

Remarque : il faut toujours utiliser les parenthèses pour indiquer les groupements exacts constituant des expressions. Ceci accroit la lisibilité du code et évite des erreurs de programmation.

7.3.4 Expressions String

Le seul opérateur utilisable avec des expressions de type "string" est le signe +.
Il permet de regrouper plusieurs expressions "string". Exemple:
"Hello" + "World!" donnera une nouvelle expression string : "HelloWorld!"

7.4 Déclaration des Variables

Toutes les variables utilisées dans un programme Express doivent être déclarées dans Variable Declaration Section, c'est-à-dire dans la section Déclaration des Variables. On précise le nom et le type de la variable.
Le nom doit toujours commencer par une lettre; ex : numeric weight.

On peut indiquer la valeur initiale de la variable par un chiffre entre parenthèses après le nom; ex : numeric weight (0.5);

Si plusieurs variables sont de même type, on peut les positionner à la suite les unes des autres, séparées par des virgules. Ex avec 3 variables de type numérique : numeric weight, factor, delta;

Conseil : utilisez toujours des noms pour vos variables qui donnent leur signification. Ainsi, le code sera plus facile à comprendre plus tard.

Les types de variables sont : numeric, bool, series et string.
Si la valeur initiale n'est pas précisée dans la& déclaration, Express prend, par défaut, les valeurs suivantes :

Type	Valeur initiale
numeric	0
bool	false
series	tous les éléments à 0
array	tous les éléments à 0
string	" ", c'est-à-dire string vide

Une serie peut être liée à une série d'un autre Express sentimentor ou déjà dans la liste des sentimentors de base. Se reporter à la section "Importer une série d'un autre sentimentor".

Exportation de Variables

Les variables numeric and string peuvent avoir une clause facultative export:
export “label;format”;

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

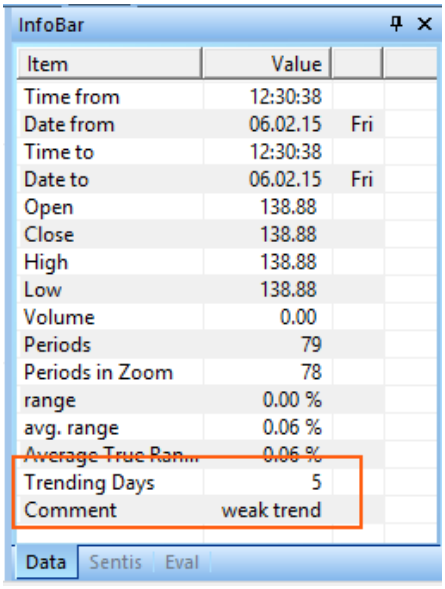
La partie “;format”-est facultative. (Voir fonction NumericToString() pour une description de
string au bon format.)

Les variables exportées sont affichées dans des colonnes à part dans le tableau des résultats
des Screeners:

Screeners

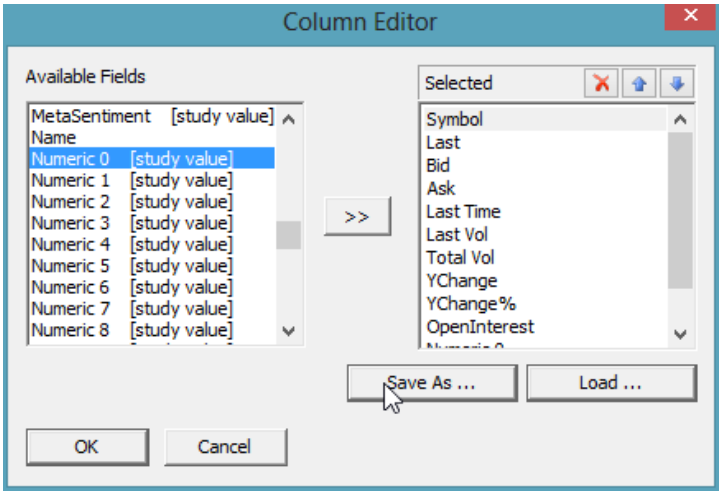
Name	Trending Days	Comment
Carrefour	11.00	strong trend
Bank of America	5.00	weak trend

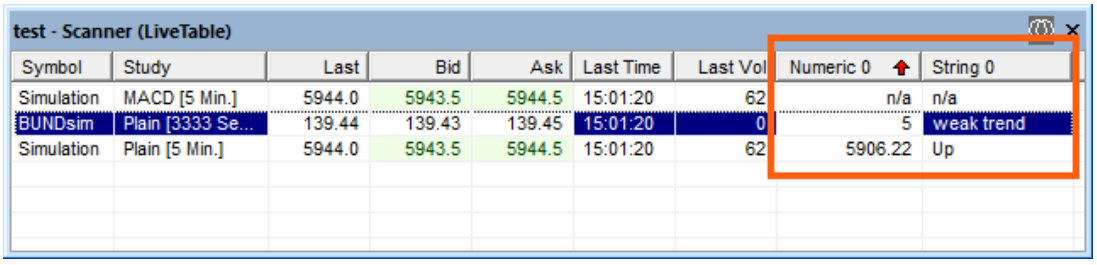
Les variables exportées se trouvent aussi dns la Barre Info de l’étude:



La fenêtre de configuration pour les Scanners (LiveTables) permet de montrer les variables
exportées des scripts Express.

Dans la mesure où une LiveTable peut contenir diverses études, l'entête de colonne ne peut
être que générique. On peut y voir jusqu'à 10 variables Numeric/Series exportées et 10
Strings exportés:





7.5 Les variables d'Entrée (Input Variables)

Les entrées, ou Input, sont des variables dont la valeur est déterminée par l'utilisateur.
Elles apparaissent dans la Barre de Personnalisation en tant que paramètres, et l'utilisateur peut les modifier, comme c'est le cas pour tout paramètre des sentimentors de base.
Ces variables peuvent faire l'objet d'un processus d'optimisation. Il est donc nécessaire de préciser la valeur minimale, la valeur maximale, et la valeur initiale.

La syntaxe de déclaration d'une variable input est la suivante :

Input $<var name>(<min value>, <max value>, <initial value>);

Exemple :
input $span(1,200,20);

Le signe $, utilisé en tant que préfixe, indique , où que ce soit dans le code, qu'il s'agit d'une variable input qui est sujette à optimisation, et revêt par conséquence une grande importance pour le calcul global.

Très souvent ces valeurs sont exprimées seulement en nombre entier, mais il arrive cependant qu'il faille utiliser des nombres décimaux. Deux spécifications supplémentaires sont alors nécessaires: la précision et le pas. Le pas représente la taille minimum de tout changement lors du processus d'optimisation.
La déclaration d'un variable input exprimée en chiffres décimaux est la suivante :
Input $<var name> (<min value>, <max value>, <initial value>,<step size>, <precision>);

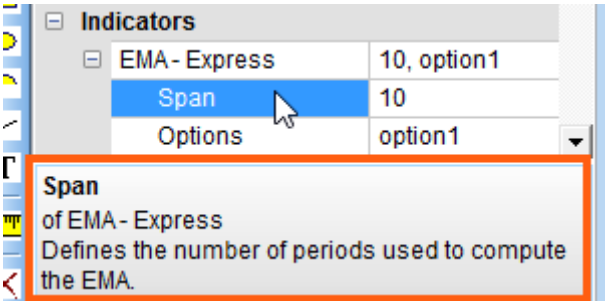
Exemple:
input $factor(1.00, 3.00,2.00,0.01,2);

Showing Explanations

An input variable can have an optional info clause. E.g.:

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

When clicking the parameter in the DesignerBar, the text is displayed in the Description Area of the DesignerBar:



This allows to have a short description of each input parameter so the user does not need to refer to some external documentation.

7.6 Accéder aux Variables et aux Données Séries

On accède à la valeur d'une variable tout simplement en utilisant le nom de la variable, comme dans :
factor*delta
factor et delta étant des variables de type numérique.

Une variable de type serie fait toujours référence à la bougie en cours. La syntaxe est la suivante:
<series name> [n]
n désignant un nombre entier positif ou une expression. Exemple :

close[1] or close[span], span étant alors une variable numérique.

L'exemple prend en compte la valeur de la barre précédente (1 barre en arrière).

Pour accéder à la barre en cours, on peut utiliser la notation close au lieu de close[0].

7.7 Utiliser les Arrays

Un array est un tableau comportant un nombre fixe d'entrées de type numeric.
Un array est défini dans la Section Déclaration de Variables comme suit :
Array a(100)
Le chiffre 100 indique le nombre d'entrées numériques contenues dans l'array.
Il y a une différence importante de notation entre les entrées array et les entrées series, bien que la syntaxe soit la même :
<array name>[n]

Dans une série, close par exemple, close[n], indique le prix de clôture n barres en amont de la barre courante, et close[0] indique le prix de clôture de la barre courante. De ce fait, la barre correspondant, par exemple, à close[20] se déplace d'un cran chaque fois qu'une nouvelle barre est initiée.

Mais dans array, a[0] fait référence à la première entrée dans l'array, et a[n]correspond à la valeur de l'array située n barres plus loin. A[20] correspondra donc toujours à la même barre, et est indépendante de la barre courante.

Note : du fait que a[0] correspond à la première barre, la numérotation d'un array a[100], c'est-à-dire comprenant 100 entrées , ira de 0 à 99.

Un array peut être redimensionné en utilisant la fonction *SetArraySize(),*
par exemple *SetArraySize(a,250);*

Toutes les entrées d'un array peuvent être réglées à une valeur déterminée en utilisant la fonction *SetArrayTo()*
Par exemple *SetArrayTo(a,500);*

7.8 Assigner une valeur

Une valeur déclarée peut se voir assigner une valeur en utilisant l'opérateur =.

Span = high – low : on assigne la valeur High-low à la variable Span

Lorsque l'on affecte une valeur à une variable de type series, Express utilise l'élément courant traité dans la section calcul : si, par exemple, median est une variable de type series et que le calcul soit fait pour la 25ème barre du graphique, alors
Median = (high + low + close) / 3
correspondra au résultat de l'expression pour le 25ème élément de la serie median.

7.9 Assigner un sentiment

Les sentiments d'un sentimentor programmé dans Express sont stockés dans les series prédéfinies sentiment.
Un sentiment devant avoir une valeur comprise entre 0 et 100, Express attribue la valeur 100 lorsque la valeur du sentiment dépasse 100, et la valeur 0 lorsque le sentiment est négatif.

Lors du calcul d'un sentiment, on peut vouloir attribuer une valeur non seulement pour la période en cours, mais aussi, par exemple, pour les deux périodes suivantes. On utilisera pour cela la syntaxe suivante :

Sentiment = [100; 90; 80];
Avec cette fonction la première période obtient la valeur 100, la suivante 90 et ainsi de suite.
Note : cette "liste des assignements" n'est valide que pour les séries prédéfinies sentiment.

7.10 Séries Prédéfinies

Les séries suivantes sont prédéfinies :

Open, close, high, low, volume. On peut aussi utiliser les abréviations :
O, c, h, l, v.

Pour les graphiques Renko, les séries renkoHigh et renkoLow sont disponibles pour accéder au haut et au bas d'une brique.

Sont également prédéfinies les séries suivantes :

Séries	**Signification**
date	la date de fin de période
dateOpen	la date de début de période
time	l'heure de fin de période
timeOpen	l'heure de début de période
dateTime	les date et heure de fin de période
dateTimeOpen	les date et heure de début de période

En résumé, les séries sentiment sont utilisées pour stocker les sentiments qui ont été calculés, sauf si on utilise un schéma d'interprétation par défaut.

A l'exception des séries sentiment, toutes les séries prédéfinies sont en "lecture seulement", il n'est pas possible de leur affecter une valeur.

7.11 Importer une Série depuis un autre Sentimentor

Une variable series peut être associée à une series d'un autre Express Sentimentor, ou à une series d'un sentimentor de NanoTrader, en tant que partie de l'étude en utilisant la syntaxe suivante :

series myseries (sentimentorName.seriesName);

La variable myseries fait référence, en lecture seulement, à la série nommée seriesName du sentimentor nommé sentimentorName.

Ce schéma est très utile lorsqu'on utilise le résultat du calcul fourni par un sentimentor qui génère des signaux dans un autre sentimentor utilisé en tant que stop.

Exemple:

```
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);
```

Le sentimentor EMA Mid génère des signaux sur la base d'un prix de déclenchement calculé à partir des EMAs des highs and lows.

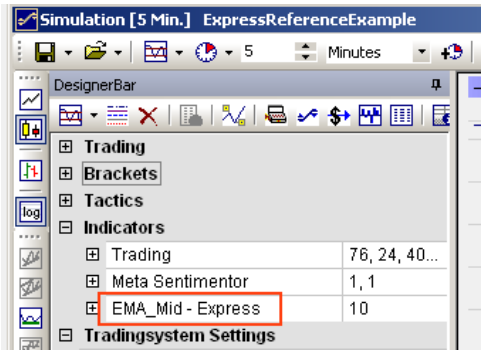
Supposons maintenant un sentimentor Stop
Exemple ci-dessous :

```
Express Stop EMA_MidStop
vars
input $ofs (-25, 25, 10);
series anchor (EMAMidExpress.emaMid);
//le Stop suit la série importée nommée 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
```

Règles d'écriture des noms

Le sentimentor importé doit s'écrire de la même façon que dans la Barre de Personnalisation en omettant les caractères non alphabétiques et les espaces.

Par conséquent " EMA_Mid_Express " s'écrira EMAMidExpress. On aurait aussi pu écrire EmamidExpress, puisque la casse utilisée n'a pas d'importance.



Si l'étude comprend plusieurs "EMA_MidExpress", ceux-ci doivent être indexés :

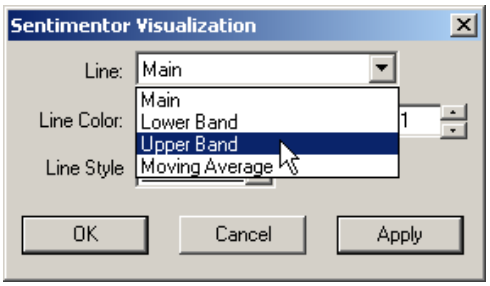
series anchor (EMAMidExpress**2**.emaMid);

Importer depuis un sentimentor standard

On peut importer une série depuis un sentimentor standard, ex:

series myBollinger (BollingerBands.upperBand);

Les séries qui peuvent être importées sont listées dans la boite de dialogue Visualisation du sentimentor. Pour les bandes de Bollinger, celle-ci se présente comme suit :

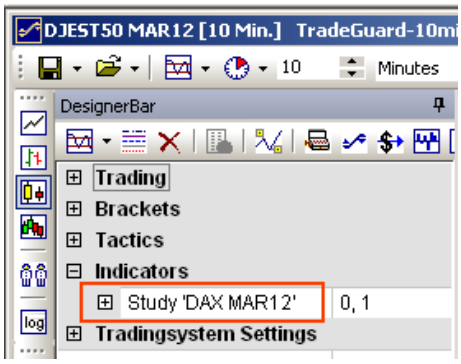


Importation en cascade

Si un sentimentor A importe une série depuis un sentimentor B, A devra être recalculé chaque fois que le paramétrage de B sera modifié.
NanoTrader déclenche cette séquence automatiquement.
Il n'y a pas de limite de principe au nombre de sentimentors pouvant être importés, ceci pouvant se faire en cascade : A importe de B, le quel importe de C, lui-même important de D, etc.

7.12 Importer une Donnée de prix depuis un autre symbole

Comme le cas particulier “importer une série depuis un autre sentimentor” montré ci-dessus, les données de prix d’un symbole utilisées par le sentimentor Study sont accessibles.
Rapellez-vous qu’une des utilisations du sentimentor Study est simplement d’afficher les données de prix du symbole appelé.
Supposons qu’un sentimentor Study appele le future DAX:



Un sentimentor Express pourrait appeler les données de prix de la façon suivante:
series dax (StudyDaxMar.close); //or .open, .high, .low
Comme les contrats Future ont leur échéance à la fin de leur nom, ce qui nous force à adapter le code à chaque expiration, il est possible de ne donner que le début de la séquence du nom du contrat, par ex. :
series dax (StudyDax.close);

7.13 Constantes de date et d'heure

Le format **date** est: DD_MM_YYYY
Exemple : 22_10_2010

Le format **heure** est : HH:MM ou HH:MM:SS. Exemples 14:15 , 14:15:40
Si les secondes ne sont pas indiquées, elles sont considérées comme valant 0.

Les constantes date et heure sont utilisées dans les expressions booléennes, ex:
if time < 10:00 then
  sentiment = 50; //pas d'achat pendant la 1ère heure de session
else …

7.14 Les Instructions

Les instructions sont constituées de mots réservés, d'opérateurs et d'opérants; Elles se terminent par un point virgule. Ex :
delta = high – low;
MovingAverage (close, mySeries, 10);
if (open > close) then up = up + 1 end

7.15 Structures de contrôle

7.15.1 if..then

La structure conditionnelle "if … then..)" (si…alors…) s'utilise lorsqu' une condition spécifique est requise pour exécuter une instruction. La syntaxe est:

if <boolean expression> then
  <statement>;

Exemple:

```
if close > open then
  upMoves = upMoves + 1;
```

Si plusieurs instructions doivent être exécutées, la syntaxe suivante doit être utilisée:

```
if <boolean expression> then
begin
  <statement>;
  <statement>;
  ...
  <statement>;
end
```

7.15.2if..then..else

L'expression "if.. then else" (si…alors …à défaut) s'utilise lorsqu'une instruction doit être appliquée lorsque la condition *n'est pas* réalisée. Ex :

```
if <boolean expression> then
   <statement>;
else
   <statement>;
```

Lorsque plusieurs instructions doivent être réalisées, il faut encadrer celles-ci par les mots "begin" et "end" (début et fin). Ex:

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

7.15.3Boucle conditionnelle While (tant que)

La syntaxe d'une boucle while est la suivante :

```
while <boolean expression>
begin
  <statement>;
  <statement>;
...
```

```
  <statement>;
end
```

Les instructions (statements) sont exécutées tant que la condition reste valide (=tant que l'expression booléenne reste vraie).

Exemple:

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

Note: vérifier soigneusement que l'expression booléenne finira par devenir "fausse"; à défaut la boucle While sera une boucle sans fin, et le processus se bloquera. NanoTrader ne peut pas vérifier le caractère fini d'une expression booléenne. En conséquence si une boucle While ne se termine pas dans un délai de 5 secondes, Nano Trader met fin au programme Express.

7.15.4Boucle itérative For

La syntaxe d'une boucle for est la suivante :

```
for <variable> = <start value> to <numerical expression>
begin
  <statement>;
  <statement>;
...
  <statement>;
end
```

La "variable" va prendre successivement toutes les valeurs de la série allant de la valeur initiale (start value) à la valeur finale, indiquée ci-dessus par "numerical expression". A chaque fois on appliquera les instructions (statements) comprises entre begin et end.

Exemple :

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

Dans cet exemple, on part de "upMoves"=0, et on fait varier la variable i de 0 à 9. La condition if close>open définit une bougie croissante. Chaque fois qu'on rencontre une bougie croissante, le compteur upMoves est incrémenté de +1 unité. Ce programme permet donc de calculer le nombre de bougies croissantes (bougies "vertes") sur les 10 dernières bougies (de 0 à 9).

On peut parfois vouloir procéder en ordre décroissant depuis la valeur initiale (Start value) jusquà la valeur finale (numerical expression), à condition, bien sûr, que le "rang" de la valeur initiale soit plus élevé que le rang de la valeur finale (par ex , de 9 à 0 au lieu de 0 à 9)
dans ce cas on utilisera la variante suivante :

```
for <variable> = <start value> downto <numerical expression>
begin
  <statement>;
  <statement>;
...
  <statement>;
end
```

On utilise for ..downto, au lieu de for…to, lorsque la valeur initiale est plus grande que la valeur finale.

Note: vérifier soigneusement que la variable peut effectivement dépasser la valeur "numérical expression (ou tomber en dessous, en cas de downto). NanoTrader ne peut pas vérifier le caractère fini de la série for ..to. En conséquence si le processus ne se termine pas dans un délai de 5 secondes, Nano Trader met fin au programme Express.

7.16 Exécution rapide des programmes

Lorsque vous utilisez des boucles While ou For veillez à ce que le calcul ne consomme pas trop de ressources machine. On peut parfaitement implanter un mode de calcul naturel, et qui est exact, mais qui va demander des ressources beaucoup plus importantes qu'une autre solution, et donc ralentir tout le processus.

Prenons l'exemple d'une moyenne mobile de 50 périodes. L'approche naturelle est de faire la somme des prix de clôture des barres 0 à 49, et de diviser le résultat par 50. Il faut donc, à chaque barre, additionner 50 valeurs, et diviser le résultat par 50. Une approche plus simple consiste à remarquer que la somme pris en compte pour la barre suivante sera la somme obtenue précédemment, majorée du prix de clôture de la nouvelle barre et diminuée du prix de clôture de la barre la plus ancienne (la plus à gauche, la 'leftmost bar"). On a, avant la division par 50, 1 addition et 1 soustraction, au lieu de 49 additions! En gros, cette méthode est 50 fois plus rapide que l'approche naturelle.

Pour une MM de 200 périodes, on serait 200 fois plus rapide! Imaginez ce que cela signifie en temps de calcul si on est dans un processus d'optimisation.

Dans Express, l'utilisation de la méthode décrite ci-dessus, se traduit par l'utilisation de la fonction booléenne IsFinalBar() :

```
series result;
input $span (1, 200, 10);
Calculation:
..result = ...;
  if IsFinalBar() then  //true, if currently the final bar is processed
      MovingAverage (result, result, $span);    //built-in function
```

Chaque fois que vous supposez que votre script requiert un temps de calcul important, vérifiez que, au début du script, vous avez "CalculateAtEveryTick(false);". De la sorte le calcul ne sera exécuté qu'en fin de période, et non à chaque tick, ce qui diminuera très significativement les temps de calcul.

7.17 Interprétation – Calcul des sentiments

L'objectif premier d'un sentimentor est le calcul d'un sentiment pour chaque période. Ceci se fait via la section Interpretation du programme Express. Cette section débute toujours par le mot clé Interpretation.

7.17.1 Interprétation dans le cas des systèmes pré-établis

Dans un grand nombre de cas, le calcul des sentiments peut se faire directement par l'un des schémas utilisés avec les sentimentors préétablis (Se reporter au manuel "Systèmes de trading , chapitre 5). Lorsqu'on utilise un schéma préétabli, l'éditeur correspondant au schéma concerné est disponible, indiquant les détails et permettant de modifier éventuellement les paramétrages. Se référer à un système préétabli simplifie grandement la programmation d'Express sentimentor.

Un schéma préétabli répond aux règles habituelles de fonction.
Exemple :

interpretation TwoThresholds (mySeries, $upZone, $downZone);
ou
interpretation TriggerLine (close, mySeries);
Le schéma TriggerLine calcule les sentiments à partir du croisement des cours de clôture et de la mySeries (par ex une MMA).
Dans le cas où un schéma pré-établi nécessite de préciser des valeurs de paramètres, celles-ci sont données dans input variables

Les schémas préétablis suivants sont disponibles :

Schéma pré-établi	**Utilisation type**
TwoThresholds (series curve, input upThreshold, input downThreshold)	RSI
TriggerLine (series curve, series trigger)	Crossing MA
Swing (series curve, input spanLeft, input	Momentum

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

Lorsqu'on utilise un schéma préétabli, il n'est pas nécessaire de préciser les conventions de dessin (section plot); Express utilise alors celles du schéma préétabli. Il faudra cependant donner les instructions pour au moins une courbe, à défaut le mécanisme standard n'est pas appliqué.

7.17.2Programmer une Interprétation spécifique

Dans le cas où aucun schéma préétabli ne peut s'appliquer, on utilisera la syntaxe suivante :

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

Notez les mots begin et end encadrant les instructions (statements)

Le mode de calcul des sentiments est indiqué dans la section calculation , les instructions sont exécutées barre par barre, en partant de la plus ancienne (la plus à gauche).

Les sentiments sont affectés à la série sentiment. Nano Trader initialise cette série avec lea valeur 50, cet à dire position neutre.

Exemple:

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

On peut affecter un sentiment, non seulement à la barre courante, amis aussi aux barres suivantes. Cette technique permet de prendre en compte un évènement, non seulement sur la barre où il se produit, mais également sur les barres suivantes.

Exemple:

```
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
```

La listed'affectation (list assignment)
`sentiment = [value; value;...;];`
n'est valable que pour les séries sentiments.

7.18 Le Tracé

Les instructions finales d'un programme Express, relatives aux tracés, consistent en une ou plusieurs instructions, selon la syntaxe:

`plot (`<series name>`, `<colorname>`, `<pen width>`);`
ou

`plotline (`<constant or variable>`, `<colorname>`, `<pen width>`);`

Exemples:

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

Les couleurs suivantes sont prédéfinies:
`red, lightRed, green, lightGreen, blue, lightBlue, magenta, lightMagenta, yellow, lightYellow, cyan, lightCyan, grey, black, white.`
Si aucune couleur n'est precisée, la couleur "bleu" est choisie.

Une autre façon, plus large, de définir une couleur est d'utiliser les parmètres RGB (Red, Green, Blue), selon la syntaxe:

`plot (`<series name>`, `<red>`, `<green>`, `<blue>`, `<pen width>`);`

Les valeurs Red, Green, Blue sont des nombres entiers, compris entre 0 et 255, qui définissent l'intensité de chacune de ces 3 couleurs.

Il est parfois intéressant de tracer des bougies ou des barres, basées sur des données de prix réels ou modifiés. On utilise alors la syntaxe suivante:

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

ou

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

Pour remplir l'espace entre deux séries, utiliser plotband:

```
plotband (<upper series name>, <colorname>, <pen width>,
          <lower series name>, <colorname>, <pen width>,
          <fillcolor>);
```

Si deux séries croisées doivent être affichées avec un remplissage en couleur utilisez `plotcrossinglines` (par ex. pour des sentimentors comme le Croisement de Moyenne Mobiles) :

```
plotcrossinglines (<series1 name>, <colorname>, <pen width>,
                   <series2 name>, <colorname>, <pen width>,
                   <fillcolor series1 above series2>,
                   <fillcolor series1 below series2>);
```

8 Exemple d'un sentimentor bloqueur

La valeur d'un sentiment est comprise entre 0 et 100. Nano Trader accepte également deux états complémentaires, utilisés en conjonction avec des filtres:

- BLOCK: les signaux, Longs ou Courts, sont refusés
- FLAT : les signaux, Longs ou Courts, sont refusés, et toutes les positions ouvertes sont fermées

Lorsqu'on utilise des sentimentors manuels, l'utilisation de ces états permet d'interdire le trading à certaines heures.

On peut également les programmer en Express.

L'exemple suivant est relatif à un Stop bloqueur basé sur le volume:

```
Express Blocker VolCheck                                    1.
Vars
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;  //just for illustration       3.
  else
    sentiment = senti_pass;                                 4.
end

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

Explications:

1. Le mot clé Blocker indique que ce sentimentor va fonctionner en tant que bloqueur.

   Ceci autorise l'utilisation des constantes senti_block, senti_flat et senti_pass. En outre, ces mots clé limitent l'utilisation du sentimentor à la fonction filtre.

2. L'utilisation de la constante senti_block fait que tout signal, Long ou Court, est refusé (bloqué)
3. L'utilisation de la constante senti_flat fait que tout signal, Long ou Court, est refusé (bloqué), et que les positions ouvertes sont fermées.
4. Avec senti_pass aucun signal n'est filtré

9 Exemple d'un sentimentor Stop

L'exemple ci-dessous illustre la méthode pour implanter un sentimentor Stop :

```
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.
```

Explications:

1. Le mot clé Stop indique que le sentimentor va fonctionner en tant que stop basé sur le prix. Certaines fonctions ne seront alors disponibles que pour ce type de sentimentor. Il pourra être ajouté en tant que stop à une étude.
2. La fonction MarketPosition()renseigne sur la position courante :
   1 = Long
   0 = Flat
   -1 = Short
3. La fonction booléenne IsIntradayEntry() est true (vraie) si la position vient d'être ouverte dans la barre courante, la période n'étant pas encore terminée.

Il est parfois nécessaire d'utiliser, pour la première période, un schéma de calcul différent de celui des périodes suivantes, le calcul du stop se faisant alors –par exemple- sur la seule base du prix d'entrée. Si la position n'est pas fermée au cours de cette période initiale, le prix du stop pour la période suivante sera recalculé.

4. La fonction SetStopPrice() positionne la valeur de calcul du stop dans Nano Trader qui choisira le stop le plus serré parmi ceux figurant dans l'étude.
5. Pour les sentimentors Stop, il n'y a ni interprétation ni dessin, donc pas de Interpretation Section, ni de Plot Section .

10 Exemple d'un Stop / Tactic avec adaptations intermédiaires

Lorsqu'on active le sentimentor Stop, le stop se place $initialRisk ticks en dessous du prix d'entrée ($initialRisk est un nombre entier qui définit, en nombre de ticks, le risque initial).
Lorsque le cours atteint entry price + $profitTrigger ticks, le stop se déplace et s'ajuste à entryPrice + $initialProfitOffset.
A partir de là, le stop se démplace en respectant un écart avec le cours de $trail ticks. Si le paramètre $trail est réglé sur 0, le stop n'est pas suiveur.

```
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
    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
    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
```

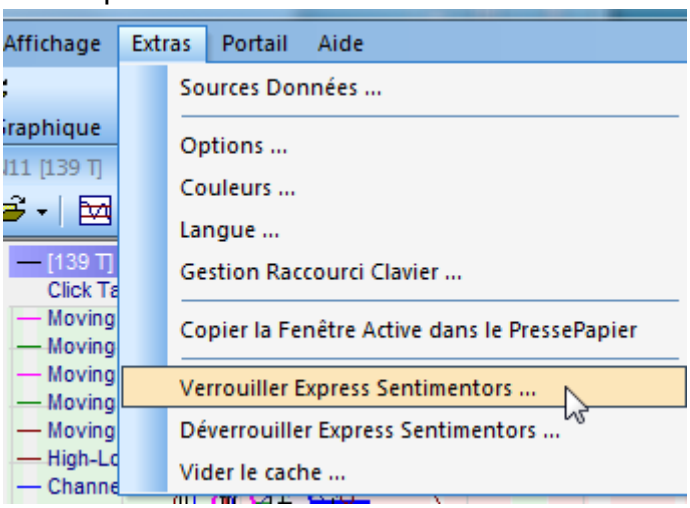
Explications:

1. L'existence de cette fonction dans le code active les adaptations intermédiaires
2. Le prix d'ouverture de la position, tel qu'il apparaît dans le compte
3. Le prix le plus haut atteint dans la barre d'ouverture postérieurement au prix d'entrée.
   Ex : supposons un graphique en périodes de 60 mn. Si l'entrée en position se fait au bout de 25 mn, MaxPriceEntryBar()est le plus haut des 35 mn restantes.

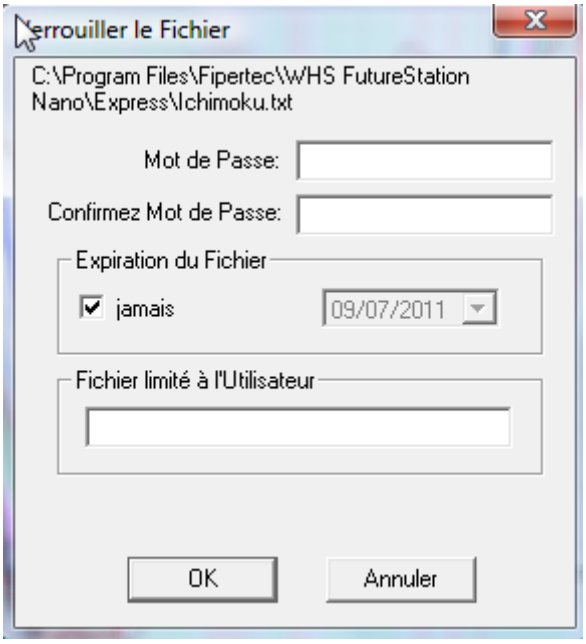
11 Crypter Express sentimentor

Les sentimentors express peuvent être cryptés. Ceci ouvre un large éventail d'applications commerciales. L'utilisation d'un sentimentor crypté est exactement la même que celle d'un sentimentor standard. Cependant le code lui-même ne peut être ni vu ni édité par l'utilisateur.

Pour crypter un Express sentimentor, cliquez, dans le menu principal du graphique sur Extras/verrouiller Express Sentimentor :



Cette action fait apparaître la liste des sentimentors Express. Sélectionnez celui que vous voulez crypter. La boîte de dialogue ci-dessous s'affiche :



Vous pouvez:

- Définir, et si besoin, modifier un mot de passe, lequel sera nécessaire à l'utilisateur
- Limiter la période d'utilisation du fichier (ou, au contraire –case "jamais"- ne pas la limiter)
- Limiter l'utilisation du fichier à un utilisateur nommément désigné.

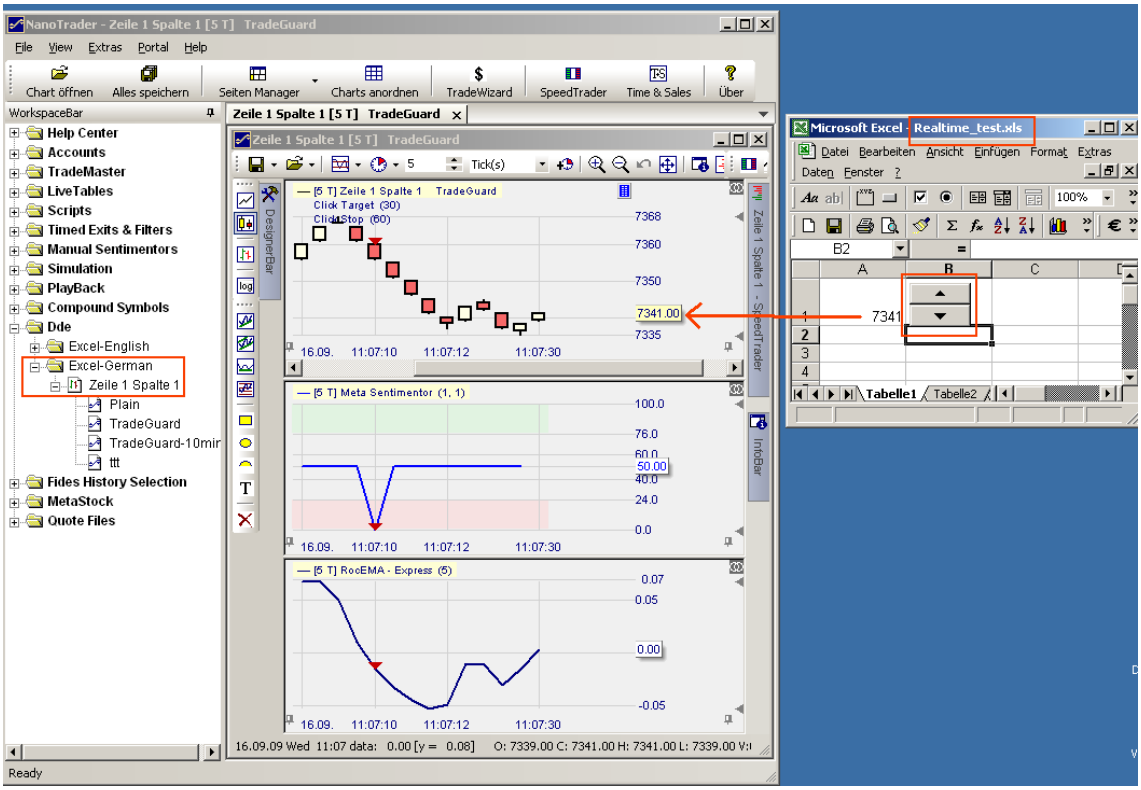
Lorsque vous mettez à disposition une étude complète comprenant un Express sentimentor crypté, il est nécessaire de donner également l'accès à l'Express sentimentor crypté.

12 En cas de "Bug"

Express est conçu pour des développements assez légers. Il ne contient pas d'outil spécifique pour traiter les bugs (blocages de programme). Différentes techniques peuvent vous aider:

- Tracer une série qui comprend des valeurs intermédiaires
- Utiliser les fonctions Annotate() ou Highlight() pour dessiner les valeurs et supprimer les bugs des notes directement dans le graphique
- Utiliser SowTip() pour attribuer des valeurs et des notes aux barres individuellement.

Il est souvent nécessaire de tester un environnement spécifique de données. Il est conseillé de créer les données via Excel. Pour cela, activer DDE dans Extras/Datasources. Dans le réperoire de NanoTrader figure le dossier *Realtime_test.xls*. Ouvrez-le et ajoutez le sentimentor qui doit être "dé-bugger" à une étude basée sur Excel. 5selon votre version d'Excel, vous pourriez avoir besoin d'utiliser un symbole disponible dans le dossier"Excel-English".)



13 Fonctions et procédures disponibles

Les fonctions

Express met à disposition un certain nombre de fonctions prêtes à être utilisées dans un programme Express. Si ces fonctions nécessitent des paramètres, Nano Trader vérifie que les paramètres correspondent à la "définition de fonction". Une "définition de fonction" précise le nom de cette fonction.
Exemple : la définition de fonction de Max()est donnée par :

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

Cette fonction renvoie des valeurs de type float (décimal). Elle prend en compte deux paramètres, tous deux de type float . Rappelons que, si nécessaire, Express convertit en float les nombres entiers, ainsi

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

est une formulation admissible, car 5000 sera automatiquement converti en valeur décimales.

Les fonctions qui retourne comme résultat une valeur sont utilisables dans des expressions telles que mySeries = Max (open, close) * 2;

Les procédures

Une fonction qui ne retourne pas une valeur est appelée une procédure. Une procédure ne peut pas être utilisée dans une expression. Elle forme une instruction complète:

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

La définition de MovingAverage est:

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

La terminologie void indique qu'il n'y a pas de valeur en retour.
Les noms des paramètres dans les définitions sont choisis de telle sorte qu'ils rappellent leur rôle dans la fonction.

Même lorsqu'une fonction ne comporte aucun paramètre, les parenthèses () doivent être maintenues. Ex:

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

Liste des fonctions disponibles :

Définition: float **AbsValue** (float value)

Signification: Retourne la valeur absolue de `value’.

Exemple: AbsValue (-3.7) retourne 3.7; AbsValue (5) retourne 5

Définition: 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)

Signification: Ajoute une annotation à la barre en cours en fonction du ‘type’ choisi et dans la ‘couleur’ spécifiée.

Les types suivants sont pris en charge : (voir capture d’écran)

“ellipse”, “upTriangle”, “downTriangle” ainsi que “labelLeft”, “labelCenter” et “labelRight”.

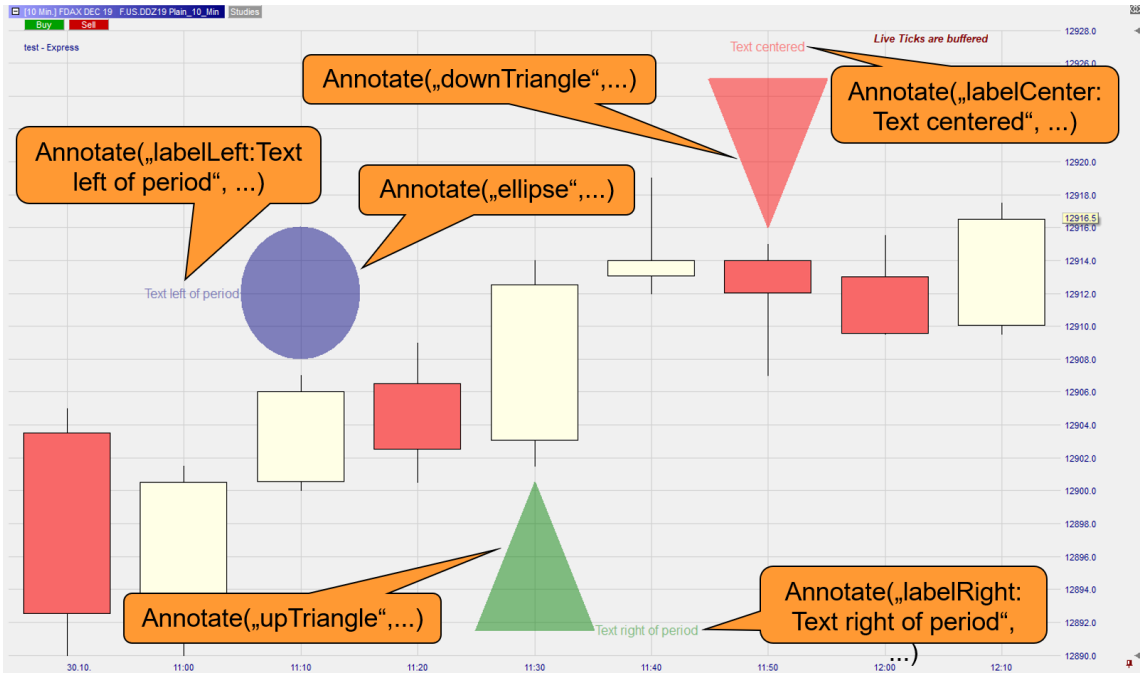
La position verticale et la taille de l'annotation pour les types "ellipse", "upTriangle" et "downTriangle" est définie par les paramètres "high" et "low".

Le texte à afficher avec "labelLeft", "labelCenter" et "labelRight" est ajouté au type en le séparant par deux points, par ex :

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

Un saut de ligne peut être inséré en utilisant la séquence \n. Les lignes ne sont pas enveloppées automatiquement.

La position verticale du texte est définie par le paramètre "high".



Voir la fonction "plot" pour la liste des noms de couleurs supportés.

Plusieurs annotations peuvent être superposées.

Exemple:
if (volume > 500) then Annotate(“ellipse”, “green”, high, low);

if (volume > 500) and IsBarCompleted() then
    Annotate(“ellipse”, “green”, high, low); //no intra bar annotation

Les versions "At" permettent d'annoter la barre courante, comme pour l'indexation des données de prix. Ex. utilisez AnnotateAt(2, "upTriangle", "blue", high, low) pour placer une annotation deux points avant la barre courante. Il est ainsi facile d'annoter des modèles de prix qui, par exemple, s'étendent sur plusieurs périodes.

Voir aussi Highlight().

Définition: float **ArcTangent** (float value)

Signification: Retourne the arcus tangent of `value’.

Exemple: ArcTangent (2.7475) retourne 70.

Définition: float **Atr** (int span)

Signification Retourne Average True Range du graphique principal pour la barre courante et les précédentes "span" barres.

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

Signification: Retourne Average True Range du graphique principal pour la barre courante et les précédents ‘span” barres exprimées en points.

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

Signification: interprétation standard de schemas où les sentiments sont calculés pour des séries "upper" (haute) et lower (basse)

Exemple: interpretation Bands (close, myLower, myUpper);

Définition: int **BarsSinceEntry** ()

Signification: Nombre de périodes depuis l'ouverture du trade.

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

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

Cette fonction n'est valable que pour les stops sentimentors.

Signification: CalculateAtEveryTick(false) empèche l'exécution d'un script à chaque nouveau tick. Le script ne sera exécuté qu'en fin de période. Ceci réduit considérablement la consommation de ressources et accélère d'autant le programme.

Exemple: if IsFirstBar() then CalculateAtEveryTick(false);

Définition: float **Ceiling** (float value)

Signification: retourne l'arrondi supérieur en nombre entier de `value'

Exemple: Ceiling (2.95) retourne 3

Définition: float **Cosine** (float value)

Signification: retourne le cosinus de `value' exprimé en degrés.

Exemple: Cosine (45) retourne 0.7071

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

Signification: crée un fichier ayant comme contenu le texte donné. Si le fichier existe déjà, le texte donné y sera ajouté. Le chemin et le nom du fichier sont définis par le fichier de paramètres.

Voir la fonction PlaySound() pour une description des conditions dans lesquelles un fichier est créé. Les mêmes principes que pour l'émission de sons s'appliquent.

Exemple:
```
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());
```

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

Signification: retourne true (if curve[1] <= trigger [1]) and (curve > trigger), false dans le cas contraire

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

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

Signification: retourne true (if curve[1] >= trigger[1]) and (curve < trigger), false dans le cas contraire

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

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

Signification: Retourne true (if curve[1] >= threshold]) and (curve < trigger), false dans le cas contraire

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

Définition: int **CurrentBarIndex** ()

Signification: Indique l'index de la barre en cours de calcul. L'index de la première barre est 0.

Exemple: highDiff = CurrentBarIndex() – IndexOfHighest (close, 10);

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

Signification: convertit une date en valeur numérique. La date 2013-04-23 est convertie en 130423.

Cette fonction est très utile lorsque le calcul se base sur une date.

Exemple:

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

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

Signification: durée, en secondes, du début à la fin.

Exemple: d = Duration (timeOpen, time);

d = Duration (dateTime[20], dateTime);

Note: Le programme doit s'assurer que les deux paramètres sont de même type. Par exemple Duration(dateOpen, time) donnera un résultat incorrect si "DateOpen" ne contient que la composante jour, alors que "time " ne contient que la composante heure.

Définition: int **DayOfWeek** (time time)

Signification: index du jour de la semaine avec lundi=1, mardi =2,

Exemple: if DayOfWeek(date) = 5 then sentiment = 50;//no entries on Fridays

Définition: float **EntryPrice** ()

Cette fonction n'est disponible que pour les stops sentimentors.

Signification : prix d'entrée de la position en cours. Si le stop est utilisé avec TradeGuard ou avec Tactic, la fonction retourne la valeur du cours au moment de l'activation de TradeGuard ou de Tactic. Si TradeGuard était déjà actif au moment de l'entrée en position, le prix est celui de l'exécution de l'ordre.

Voir également ci-dessous: EntryPriceOriginal().

Exemple: SetStopPrice(EntryPrice() – 5 * TickSize());

Définition: float **EntryPriceOriginal** ()

Cette fonction n'est disponible que pour les stops sentimentors

Signification: le prix d'ouverture réel de la position

Voir également : EntryPrice ().

Exemple: SetStopPrice(EntryPriceOriginal() + TickSize());

Définition: float **Exp** (float value)

Signification: function "exponential" appliqué à "value".

Exemple: val = exp(1.254);

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

Signification: calcule la moyenne mobile Exponentielle des span barres de “sources” et inscrit le résultat en `target’ (objectif). Cette fonction n’est utilisée que conjointement à IsFirstBar() ou IsFinalBar().

Exemple:
if IsFirstBar () then ExpMovingAverage (close, mySeries, $span);

if IsFinalBar () then ExpMovingAverage (mySeries, mySeries, $span);

Définition: int **FinalBarIndex** ()

Signification: Retourne l’index de la période finale dans la période d’évaluation.

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

Définition: **ForceFlat** ()

Cette fonction n'est disponible que pour les stops sentimentors

Signification: Force la clôture de l'ensemble de la position. Ainsi, si plusieurs stops sont utilisés, le recours à ForceFlat() ne fermera pas seulement la partie de la position courante protégée par le sentimentor Stop référant, mais la position entière.

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

Définition: float **Floor** (float value)

Signification: retourne l'arrondi inférieur en nombre entier de "value"

Exemple: Floor (2.95) retourne 2

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

Signification: Retourne le nombre d'éléments de array `arr'.

Exemple:

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

Définition: string **GetApplicationLanguage** ()

Signification: Restitution dans la langue configurée dans NanoTrader. Restitution possible vers les langues DEU, ENG, FRA, HUN, ITA, NLD et POL.

Example:

```
if GetApplicationLanguage() = “FRA” then
  MessageBox(“Ce message est en français.”);
```

Définition: int **GetDefaultOrderSize** ()

Signification: Retourne la taille de l'ordre de l'instrument pour lequel le script Express tourne. Cette fonction est seulement valable en mode experts (voir SetExpertsMode()).

Cette fonction est idéale lorsque l'on a besoin d'augmenter/diminuer la taille des ordres en fonction de la valeur du sentiment.

Exemple:

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

Définition: time **GetExpiration** ()

Signification: retourne la date d'expiration et l'heure du symbole auquel le sentimentor Express est rattaché. Si le symbole n'a pas d'échéance, la fonction a pour résultat 1er janvier 1970 à 0:00.

Exemple:

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

Définition: string **GetPriceFormat** ()

Signification: retourne le format propre au tracé du prix sur l'axe des Y du graphique principal

Exemple: SetYscaleFormat (GetPriceFormat());

Affiche les indicateurs de l'axe des Y dans le même format que celui du graphique principal.

Definition: float **GetSpreadSize** ()

Meaning: Returns the current spread of the symbol the express sentimentor is attached to.

Example: ask = close + GetSpreadSize();

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

Signification: retourne les plus hauts des éléments des series[0], ... series[span – 1]

Exemple: tenBarHigh = Highest (close, 10);
correspond au plus haut des cours de clôture des 10 dernières barres

Définition: 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)

Signification: signale (hightlights) la barre courante en conformité avec les types de signalisation et les couleurs choisies.

Les types de signalisation possibles sont (voir capture d'écran ci-dessous):

“ellipse”, “upTriangle” (triangle vers le haut), “downTriangle” (triangle vers le bas), “slot” (période), “bottomLine”(ligne inférieure), “topLine” (ligne supérieure), ainsi que “textAbove” (texte au dessus), “textBelow” (texte en dessous).

Le texte affiché avec les types “textAbove” et “textBelow” est attaché au "type" et en est séparé par une virgule.

Un renvoi à la ligne peut être inséré en utilisant la séquence de caractères `\n`. Les lignes ne sont pas encadrées automatiquement.

Exemple:

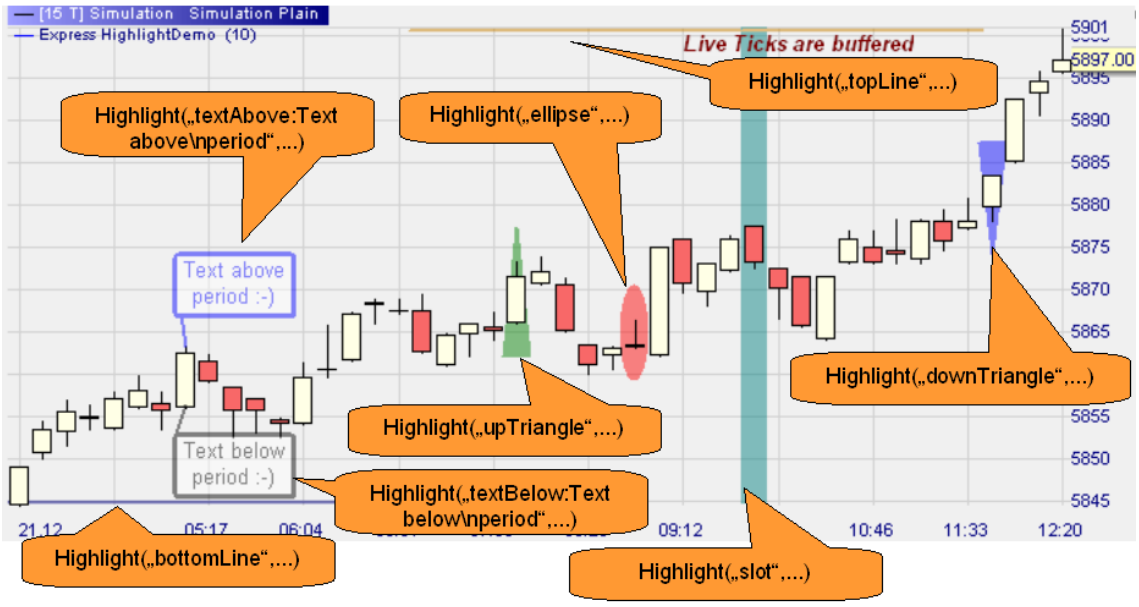
```
Highlight(“textAbove:This text appears\nabove the period”,
“black”);
```

Autre exemple :

```
Highlight(“textBelow:ALARME\nROUGE\nLOW", "Red");
```

Cet exemple renvoie, en dessous de la bougie, les 3 mots "ALARME, ROUGE, LOW" positionnés les uns sous les autres, en couleur rouge.

.



Se reporter à la fonction "plot" pour des precisions relatives aux couleurs.

Il est possible de programmer des signalisations conditionnelles:

Exemples:
if (volume > 500) then Highlight(“ellipse”, “green”);

if (volume > 500) and IsBarCompleted() then
    Highlight(“ellipse”, “green”); //no intra bar highlighting

Les versions “At” permettent de mettre en évidence la bougie en cours. Ex. utilisez HighlightAt(2, "upTriangle", "blue") pour mettre une bougie en évidence deux périodes avant la barre actuelle . Cela permet de mettre en évidence des mouvements de prix qui s’étendent sur plusieurs périodes par exemple.

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

Signification: retourne l'index du plus haut des élements series[0], ... series[span – 1]

Exemple: highDiff = CurrentBarIndex() – IndexOfHighest (close, 10);

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

Signification: retourne l'indexdu plus bas des éléments series[0], ... series[span – 1]

Exemple: lowDiff = CurrentBarIndex() – IndexOfLowest (close, 10);

Définition: bool **IsBarCompleted**()

Signification: retourne true (vrai) si la période courante est complete;

Exemple:
if IsBarCompleted() and (volume > 1000) then
  PlaySound(“gong”);

Définition: bool **IsFinalBar**()

Signification: retourne true (vrai) si la barre finale est traitée.

Exemple: if IsFinalBar () then MovingAverage (mySeries, mySeries, $span);

Définition: bool **IsFirstBar**()

Signification: Retourne true (vrai) si la première barre est traitée.

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

Définition: bool **IsIntradayEntry**()

Cette fonction n'est disponible que pour les stops sentimentors.

Signification: retourne True （vrai） si la position vient d'être ouverte dans la barre courante, la période n'étant pas encore terminée.
Il est parfois nécessaire d'utiliser, pour la première période, un schéma de calcul différent de celui des périodes suivantes, le calcul du stop se faisant alors –par exemple- sur la seule base du prix d'entrée. Si la position n'est pas fermée au cours de cette période initiale, le prix du stop pour la période suivante sera recalculé.

Exemple: if IsIntradayEntry () then SetStopPrice(EntryPrice() – 0.05);

Définition: bool **IsNewDay**()

Signification: retourne True (vrai) si la barre courante est la première barre d'une nouvelle journée ou si c'est la toute première barre des données disponibles.

Exemple: if IsNewDay () then Highlight(“slot”, “blue”);

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

Signification: retourne true (vrai) si valeur >= 0.001. Ne jamais tester avec „= 0“, car les erreurs d’arrondi font que cette condition est rarement remplie

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

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

Signification: retourne true (vrai) si valeur < 0.001. Ne jamais tester avec „= 0“, car les erreurs d’arrondi font que cette condition est rarement remplie

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

Définition: float **Log** (float value)

Signification: retourne le logarithme naturel de `value’, ou void si `value’ <= 0.

Exemple: Log (1000) retourne 6.9078;

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

Signification: retourne le plus bas des séries pour les éléments series[0], ... series[span – 1]

Exemple: tenBarLow = Lowest (close, 10);

Définition: int **MarketPosition** ()

Cette fonction n'est disponible que pour les stops sentimentors.

Signification: retourne le sens de la position en cours:
1 = long
0 = flat
-1 = short

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

Définition: int **MarketPositionSize** ()

Cette fonction n'est disponible que pour les stops sentimentors.

Signification: Détermine l'importance du volume de la position actuelle:
> 0 = long
0 = flat
< 0 = short

Les Stops incluant SetIntraPeriodUpdate() ne sont pas pris en compte par le back testing.

Exemple:
```
if MarketPositionSize() > 3 then
    SetStopPrice (low – 0.03);
else
    SetStopPrice (low – 0.01);
```

Définition: float **Max** (float value1, float value2)

Signification: retourne la valeur maximum value de `value1' et `value2'

Exemple: Max (3, 7.5) retourne 7.5

Définition: float **MaxPriceEntryBar** ()

Cette fonction n'est disponible que pour les stops sentimentors.

Signification: pour les stops avec adaptations intermédiaires (voir §10 et SetIntraPeriodUpdate().
Retourne le prix le plus haut de la période d'ouverture d'un trade, atteint *après* ouverture du trade.

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

Définition: float **MinPriceEntryBar** ()

Cette fonction n'est disponible que pour les stops sentimentors.

Signification: pour les stops avec adaptations intermédiaires (voir §10 et SetIntraPeriodUpdate().
Retourne le prix le plus bas de la période d'ouverture d'un trade, atteint *après* ouverture du trade.

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

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

Signification: affiche un `message' dans une fenêtre pop-up

Se reporter à la fonction PlaySound pour les conditions d'affichage d'un message. Les mêmes principes s'appliquent pour les messages ou alarmes sonores.

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

Définition: float **Min** (float value1, float value2)

Signification: : Retourne la valeur minimum de `value1' and `value2'

Exemple: Min (3, 7.5) retourne 3

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

Signification: calcule la moyenne mobile des span barres de "source" et inscrit le résultat en target (objectif). Cette fonction n'est utilisée que conjointement à IsFirstBar() ou IsFinalBar().

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

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

Définition: **NoDrawingOHLCinMC** ()

Signification: désactive le traçage de la série de données (par ex. barres, renko, bougies) dans le graphique principal. Cette fonction est disponible en mode experts uniquement (voir SetExpertsMode()).

Exemple: if IsFirstBar() then NoDrawingOHLCinMC();

Définition: **NormalCDF** (float value)

Signification: Retourne la valeur de la fonction densité de la distribution normale standard à valeur „valeur“.

Exemple:
cdf = NormalCDF(0.2);

Définition: **NormalPDF** (float value)

Signification: Retourne la valeur de la distribution normale standard à valeur „valeur“.

Exemple:
cdf = NormalPDF(0.2);

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

Signification: convertit "value" en une expression de temps, de sorte que "value" soit interprétée en tant que AAMMJJ, ex : 130423 est converti en date : 2013-04-23.

Si la partie "jour" de "value" est plus grande que le nombre de jour du mois en question, le dernier jour du mois est retenu.

Si la partie "mois" de "value" est plus grande que 12, la valeur de 12 est retenue.

La partie "année" ne peut être supérieure à 99

Cette fonction est très utile lorsque le calcul se base sur une date qui doit pouvoir être ajustée dans la Barre de Personnalisation et/ou l'optimiseur.

Exemple:

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

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

Signification: Formate `value' selon le format `format' et retourne le résultat en tant que string.
`Format' supporte tous les formats tels qu'ils sont utilisés pour la fonction C-function "printf()". Les formats les plus importants sont:

"%f"	Point décimal
"%6.2f"	arrondit à deux décimales
"%g"	ignore les zéros derrière la virgule
"%e"	notation scientifique

Si format est le string vide la function utilisera le format "%g".

Pour formater un prix voir la fonction PriceToString().

Exemple:
ShowTip(NumericToString(val, “%6.4f”);

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

Signification: convertit `value’ en une expression de temps de sorte que "value" soit interprétée en tant que HHMM, ex: 1545 est converti en heure : 15:45.

Si la partie "heures" de "value" est plus grande que 23, le chiffre 23 est retenu.
Si la partie "minutes" de "value" est plus grande que 59, le chiffre 59 est retenu.

Cette fonction est très utile lorsque le calcul se fait à partir de données horaires qui doivent pouvoir être ajustées dans la Barre de Personnalisation ou dans l'optimiseur.

Exemple:

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

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

Signification: joue le son référencé par "sound".
"sound" peut être un cheminement complet vers le dossier `.wav` pour être joué, ou bien il peut être le *file title* d'un dossier file logé dans le sous-répertoire Wavdu répertoire de l'installation.
Par exemple, si ce répertoire contient un mot nommé "ringing.wav", l'expression PlaySound("ringing") fait référence à ce dossier.

Un son n'est joué qu'une fois, et seulement s'il est déclenché par des données temps réel.

Note : le programme Express est exécuté à chaque tick, par conséquent, pour la période courante, le son est, par défaut, joué dès que la fonction PlaySound est activée, et non pas en fin de période.
Pour que le son ne soit joué qu'en fin de période il faut programmer:

```
if IsBarCompleted () and soundCondtion then
  PlaySound(“gong”);
```

Notez qu'un seul son peut être joué par période.
Si "sound" ne peut pas correspondre à un dossier Wav valide, un bip est émis.

Exemple:

```
if volume > 300 then
  PlaySound(“gong”);        //intra bar notification of a high volume period

if IsBarCompleted () and (close > high[1]) then
  PlaySound (“corkpop”);  //end of bar notification of a period’s close exceeding
                          //the previous period’s high
```

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

Signification: effectue le tracé des courbes des séries en utilisant les couleurs et largeurs de trait indiquées.

Exemple: Plot (close, “green”, 2);
Plot (close, 128, 128, 128, 1); //grey

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

Signification: effectue le tracé des courbes des series 1 et 2, et rempli l'espace intérieur avec la couleur spécifiée "fillColor". L'espace entre les courbes des series "upper" et "lower" sera rempli en vert clair (lightgreen). Un nombre presque infini de couleurs peut être sélectionné en utilisant la palette de couleur RGB.

Exemple: PlotBand(upper, “green”, 2, lower, “red”, 2, “lightgreen”);

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

Définition: **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)

Signification: dessine un graphique en barres en utilisant les series spécifiées. Si aucune couleur n’est spécifiée, les paramétrages du Gestionnaire de Couleur pour les barres haussières et baissières seront utilisés. Un nombre presque infini de couleurs peut être sélectionné en utilisant la palette de couleur RGB.

Exemple: 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);

Définition: **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)

Signification: dessine un graphique en Chandeliers en utilisant les séries spécifiées. Si aucune couleur n’est spécifiée, les paramétrages du Gestionnaire de Couleur pour

les bougies haussières et baissières seront utilisés. Un nombre presque infini de couleurs peut être sélectionné en utilisant la palette de couleur RGB.

Exemple: 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)

Meaning: Trace la série `curve1’ et `curve2’ et remplit l’intérieur en couleur `fillColor1’ lorsque `curve1’ est au-dessus de `curve2’. Sinon l’intérieur est remplit en couleur `fillColor2’. Un nombre presque infini de couleurs peut être sélectionné en utilisant la palette de couleur RGB.

Example: 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);

Déinition: void **PlotLine** (float value, string color, int penWidth)

Signification: trace une ligne horizontale au niveau de "value" avec les couleurs et largeurs de trait spécifiées

Exemple: PlotLine ($threshold, “red”, 2);
PlotLine (100, 150, 0, 0, 1); //dark red

Définition: float **PointValue** ()

Signification: convertit la valeur dut point du symbole auquel est rattaché le sentimentor express.

Exemple: barValue = (high – low) * PointValue();

Définition: float **Power** (float value, float exponent)

Signification: Convertit `value’ élevé au pouvoir `exponent’.

Exemple: Power (3, 3) retourne 27

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

Signification: retourne les valeurs des points Haut/ Bas/ Ouverture/ Clôture et du Volume, du jour précédent, ou retourne Void si ces données ne sont pas disponibles.

Exemple:

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

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

Signification: arrondit `value' au prix le plus proche compatible avec la taille des ticks et la précision du symbole analysé, et le convertit en string.
Prends les annotations fractionnelles en compte.

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

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

Signification : Convertit les segments de rouge, vert et bleu d'une couleur en un chiffre spécifiant la même couleur. Cette fonction est utile pour assigner des couleurs à une série en tant que paramètre pour PlotBars() et PlotCandles().

Exemple: myColors = RGB (255, 255, 160);

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

Signification: retourne `value' arrondi au nombre de décimales indiqué par `precision'.

Exemple: Round (2.428, 2) retourne 2.43

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

Signification: Retourne la "valeur" arrondie au multiple le plus proche de "multiple".

Exemple: RoundMultiple ((high + low) / 2, TickSize());

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

Signification: calcule la RSI des span barres de "sources" et inscrit le résultat en `target' (objectif). Cette fonction n'est utilisée que conjointement à IsFirstBar() ou IsFinalBar().

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

if IsFinalBar () then RSI (mySeries, mySeries, $span);

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

Signification: créé une capture d'écran du graphique principal et l'enregistre comme fichier png sous 'filename'. Si 'filename' ne contient pas de chemin complet, alors il est enregistré dans le répertoire des captures d'écran.

La capture reproduit l'affichage à l'écran à l'identique et dans les mêmes dimensions.

Il est recommandé d'utiliser plutôt la fonction ScreenshotEx() car elle offre plus de fonctionnalités.

Veuillez consulter la fonction PlaySound() pour une description du processus de la capture d'écran. Les mêmes principes s'appliquent que pour l'émission d'un son.

Exemple:
```
If IsFinalBar() then
  Screenshot(“epxress.png”);
```

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

Signification: créé une capture d'écran du graphique principale ou du Graphique Performance et l'enregistre comme fichier png dans ‘filename’. Si ‘filename’ ne contient pas de chemin complet il sera enregistré dans le répertoire des captures d'écran.

Le ‘style’ détermine les éléments du graphique principal à tracer et définit les dimensions par défaut.

0 = exactement comme affiché à l'écran
1 = moins de décorations, taille moyenne
2 = encore moins de décorations, petit
3 = afficher Equity (performance), taille moyenne
4 = afficher Equity (performance), petit

Utiliser ‘width’ (largeur) et ‘height’ (hauteur) pour définir les dimensions de l'image créée. Paramètrez à 0 pour utiliser les paramètres par défaut tel que défini par le style.

‘periodsOrDays’ définit le nombre de périodes à afficher, à compter de la dernière période disponible. Si la valeur définie est supérieure à 0 cela définit le nombre total de périodes à afficher. Si elle est égale à 0 c'est le point de départ du zoom actuel dans le graphique principal qui sera affiché. Si elle est inférieure à 0, ceci sera interprété comme des jours entiers, par ex., -2 signifie “afficher aujourd'hui et hier”.

Veuillez consulter la fonction PlaySound() pour une description du processus de la capture d'écran. Les mêmes principes s'appliquent que pour l'émission d'un son.

Exemple:
```
If IsFinalBar() then
  ScreenshotEx(“epxress.png”, 0, 600, 400, -2);
```

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

Signification: envoie un email en utilisant le sujet et le message donnés à l’adresse email configurée sous Extras/Options.

Voir fonction PlaySound() pour une description des conditions dans lesquelles un email est envoyé. Les mêmes principes que pour l’émission de sons s’appliquent.

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

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

Signification: paramètre la taille de array de`arr' à `size', c'est à dire que les éléments sont accessibles en utilisant les indices de 0 à (size – 1).

Exemple: if IsFirstbar() then arr.SetSize(arr, $arrSize);

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

Signification: paramètre toutes les entrées de array `arr' à `value'.

Exemple: if IsFirstbar() then SetArrayTo(arr, 500);

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

Signification: Paramètre la taille de l'ordre en 'valeur' de l'instrument pour lequel le script Express tourne. Comme la taille de l'ordre est modifiée par instrument, la modification s'applique à tous les comptes et études qui contiennent l'instrument. Cette fonction est seulement valable en mode experts (voir SetExpertsMode()).

Cette fonction est idéale lorsque la taille de l'ordre doit se fier à la valeur du sentiment.

Exemple:

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

Definition: void **SetExpertsMode** ()

Signification: Appelle le SetExpertsMode() pour permettre l'utilisation de cette fonction en Express qui requiert un savoir expert (Par ex. SetDefaultOrderSize()). En paramètrant le mode experts, vous confirmez que vous savez ce que vous faites.Vous êtes tenus responsables de tous les dommages causés par les fonctions expert.

Exemple: if IsFirstBar() then SetExpertsMode();

Définition: void **SetIntraPeriodUpdate** ()

Signification: Réservé aux Stop sentimentors. Le calcul du stop est fait à chaque tick. Est spécialement utilisé dans la programmation de tactics.

Les Stops incluant SetIntraPeriodUpdate() ne sont pas pris en compte par le back testing.

Note: pour des raisons internes, l'existence de la fonction call dans le code source suffit pour activer le calcul à l'intérieur de la période, même si l'instruction

correspondante n'est jamais exécutée, c'est-à-dire que, même avec un code comme "If false then SetIntraPeriodUpdate();" le calcul intra période est activé.

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

Signification: définit le prix de confirmation ou le prix limite pour un signal "Long". Pour activer l'évaluation de ce prix, l'évaluateur pour "Sentiment Signal Entrée" doit être réglé sur "Confirmation prix barre suivante" ou sur "Prix Limite barre suivante". Dans le cas où aucun sentimentor n'appelle cette routine, la confirmation de prix est réglée sur les Haut / Bas de la période ayant généré le signal.
Le prix Limite sera réglé sur le prix de clôture de la période ayant généré le signal.
Si plusieurs sentimentors appellent cette routine, le prix le plus serré est retenu.

Exemple: SetLongTrigger ((high + low) / 2);

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

Signification: analogue à celle de SetLongTrigger().

Exemple: SetShortTrigger ((high + low) / 2);

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

Cette fonction n'est disponible que pour les stops sentimentors.

Signification: Defines the stop price for the current period in case the position has just been entered or for the next period in case the position has been entered before this period.

Note: In a given sentimentor there cannot be calls to both SetStopPrice() and SetTargetPrice().

Exemple: SetStopPrice (low[-1]);

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

Cette fonction n'est disponible que pour les stops sentimentors.

Signification: définit le prix objectif pour la période courante dans le cas où la position vient juste d'être ouverte, ou pour la période suivante dans le cas dans le cas où la position a été ouverte avant cette période.

Note : pour un sentimentor donné, on ne peut pas appeler à la fois les fonctions SetStopPrice() et SetTargetPrice().

Exemple: SetTargetPrice (high[-1]);

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

Signification: définit le format de l'axe des Y en format printf

Exemple: SetYscaleFormat(“%.5d”); //utilise 5 chiffres après la virgule

SetYscaleFormat(“%g”); //utilise l’affichage compactest

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

Signification: attache un message à la barre courante. Le message est affiché dans une fenêtre pop-up lorsque le curseur est positionné sur la barre. Pour passer à la ligne suivante, faire `\n`.

Exemple: if CrossesAbove (mySeries, 70) then
Conseil d'affichage („Entrée en zone supérieure!\nAttendre confirmation.“);

Définition: float **Sign** (float value)

Signification: retourne le signe de "value".

Exemple: Sign (-3) return –1; Sign (5) retourne 1; Sign (0) retourne 0

Définition: float **Sine** (float value)

Signification: retourne le sinus de `value' degrés.

Exemple: Sine (70) retourne 0.9397

Définition: float **SquareRoot** (float value)

Signification: retourne la racine carrée de "value", ou void si "value" < 0.

Exemple: SquareRoot (4) retourne 2

Définition: void **StdDev** (series source, series target, int span)

Signification: calcule StdDev (Déviation Standard) des valeurs des séries source, source [1], ..., source[span-1], place le résultat en "target".

Cette fonction ne doit être utilisée qu'en conjunction avec IsFirstBar().

Exemple: StdDev (close, myseries, 10);

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

Signification: retourne la somme des éléments de series[0], ... series[span – 1]. Retourne void si un ou plusieurs éléments sont void.

Exemple: amount = Sum (mySeries, $span);

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

Signification: applique l'interprétation standard d'un schéma de type Swing.

Exemple: interpretation Swings (mySeries);

Définition: string **SymbolName**()

Signification: retourne le nom du symbole relatif au script en cours.

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

Définition: float **Tangent** (float value)

Signification: retourne la tangente de `value' degrés.

Exemple: Tangent (70) retourne 2.7475.

Définition: float **TickSize** ()

Signification: retourne la taille des ticks du symbole auquel Express Sentimentor est attaché

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

Définition: float **TickValue** ()

Signification: retourne la valeur des ticks du symbole auquel Express Sentimentor est attaché.

Exemple: profit = $nbTicks * TickValue();

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

Signification: convertit une expression de temps en valeur numérique. 15:45 est par exemple converti en 1545.

Cette fonction est très utile lorsque le calcul se base sur une expression de temps.

Exemple:

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

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

Signification: Formate `timeVal' selon le `format' en string
Format permet le formatage de la function C-function strftime), ex..:

%a Abrégé du nom du jour de la semaine
%A Nom du jour en entier
%b Abrégé du nom du mois
%B Nom du mois en entier
%c Date et heure locales
%d Jour du mois, de 01 à 31
%H Heure , de 00 à 23
%I Heure en format anglais, de 01 à 12

%j Jour de l'année , de 001 à 366
%m Mois, de 01 à 12
%M Minute, de 00 à59
%p Indicateur A.M / P.M. pour heure en %I
%S Seconde , de 00 à 59
%U Semaine de l'année
%w Jour de la semaine de 0 à 6
%W Semaine de l'année, Lundi étant le premier jour de la semaine, 0 à 53
%x Date en fonction de la situation géographique
%X Heure en fonction de la situation géographique
%y Année sans le siècle, 00 à 99
%Y Année avec le siècle
%z, % Nom ou abréviation du fuseau horaire, indication nulle si la zone est inconnue
%% Pourcentage

Si "format" est un string vide, time est formaté pour afficher Date et Heure

Exemple: Conseil d'affichage: (TimeToString(time, "%H:%M:%S"));

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

Signification: interprétation standard d'un schéma de croisement de deux courbes.

Exemple: interpretation TriggerLine (close, mySeries);

Définition: void **TwoThresholds** (series series, input upThreshold, input downThreshold)

Signification: interprétation standard d'un schéma de zones définies par deux seuils.

Exemple: interpretation TwoThresholds (mySeries, $upperZone, $lowerZone);

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

Signification: Si la série `source' a été importee d'un autre sentimentor qui a peut-être été agrégé, utilisez Unaggregate() pour l'afficher dans l'agrégation du MasterChart.

Exemple:

```
series maAgg(MovingAverage.main);  //le sentimentor MovingAverage est
                                   //agrégé dans l'étude
series ma;

...

if IsFirstBar() then
  Unaggregate (maAgg, ma);
```

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

Signification: calcule la moyenne mobile ponderée des span barres de "source" et inscrit le résultat en target (objectif). Cette fonction n'est utilisée que conjointement à IsFirstBar() ou IsFinalBar().

Exemple:

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