axialFINANCE

Manuel de programmation

Ariane Software

Axial Finance: Manuel de programmation

version 6.12

Table des matières

Chapitre 1. Programmation des indicateurs et signaux

1. Introduction

Ce manuel explique comment programmer les Indicateurs et Signaux personnels dans ***Axial Finance Expert*** ou ***Axial Finance Maestro***, le langage de programmation utilisé étant le *JavaScript*.

Les indicateurs et signaux programmés par l'utilisateur s'ajoutent dans la bibliothèques du logiciel qui contient déjà les indicateurs et signaux dits natifs fournis avec le logiciel.

Cette bibliothèque est consultable au menu général ***Trading System*** du logiciel :

- option ***Indicateurs***
- ou option ***Signaux***

Une fois programmé, un indicateur personnel peut être :

- affiché dans un graphique de cours
- utilisé dans le code de programmation un autre indicateur personnel
- ou utilisé dans le code de programmation d'un signal

Une fois programmé, un signal personnel peut être :

- testé dans un graphique de cours pour en voir le résultat, des flèches verticales matérialisant alors l'occurrence du signal
- utilisé dans une règle de screening
- ou utilisé pour la détection des alertes sur cours "fin de journée" ou en temps réel

Ce manuel rappelle les éléments du langage *JavaScript* utilisé pour la programmation et ensuite donne un grand nombre d'exemples concrets abordés par ordre de difficultés croissantes, chaque nouvel exemple permettant de mettre en évidence de nouvelles caractéristiques du langage utilisé.

Les lignes de programme écrites pour ***Axial Finance***, ou encore appelé le script du programme, sont indiquées dans ce manuel dans un cadre de couleur comme ci-dessous :

```
Mon script .....
```

2. Présentation

Le langage de programmation utilisé est le *JavaScript*, langage universel largement utilisé par le monde internet.

Raison de ce choix :

- D'une part, éviter un langage propriétaire ou spécifique dont la pérennité pourrait ne pas être assurée
- D'autre part, bénéficier de toutes les richesses d'un langage universel popularisé par le monde internet et qui s'enrichit en permanence

En fait, compte tenu des besoins d'***Axial Finance*** en programmation, seulement une partie de ce langage est réellement nécessaire. Une autre partie de ce langage sert aussi à l'écriture de pages web et par conséquent sans utilité dans le cas présent.

L'objet du manuel est de permettre d'acquérir les connaissances suffisantes pour les besoins de programmation les plus courants avec ce langage. Ces connaissances seront acquises progressivement à partir d'exemples concrets, exemples classés par ordre de difficultés croissantes.

Pour les utilisateurs qui souhaiteraient aller plus loin en programmation, de nombreux sites internet fournissent une description détaillée du langage *JavaScript*. Par exemple :

- http://www.laltruiste.com/document.php?url=http://www.laltruiste.com/coursjavascript/sommaire.html
- http://www.w3schools.com/js/default.asp

ATTENTION : Ces sites fournissent la totalité du langage dont certaines parties (par exemple celles propres à l'écriture de pages web) sont sans utilité pour programmer dans ***Axial Finance***. Nous conseillons à l'utilisateur de bien assimiler les notions contenues dans le présent manuel avant de consulter ces sites.

3. Définition du vocabulaire utilisé

3.1. Notion temporelle de "barre"

L'indicateur ou le signal est calculé à partir des cours de bourse présents en mémoire du logiciel. Le calcul s'effectue selon l'ordre chronologique des cours et avec la fréquence (c'est à dire l'unité de temps) du graphique sur lequel l'indicateur ou le signal s'applique (exemple 5 minutes, 1 heure, journalier, ...).

La barre définit le cours (prix d'ouverture, plus haut, plus bas, clôture et volume) pour l'unité de temps utilisé. Par exemple, on peut calculer un indicateur ou un signal en barres journalières, en barres intraday de 5 minutes, d'une heure ... Le calcul s'effectue donc pas à pas, c'est à dire barre après barre, en suivant l'ordre chronologique des cours. Le script est exécuté à chaque barre. La barre courante correspond au pas de calcul du moment.

Lors de la programmation, il n'est pas nécessaire de connaître l'unité de temps qui sera utilisée pour afficher l'indicateur dans un graphique ou tester le signal car le programme est par principe indépendant de l'unité de temps. C'est le graphique qui détermine implicitement l'unité de temps et donc la fréquence des cours à prendre en compte dans le calcul.

3.2. Le script

Le Script est le programme écrit en langage *JavaScript*.

> **Important**
>
> Les indicateurs ou signaux natifs fournis avec le logiciel ne possèdent pas de script consultable par l'utilisateur car pour des raisons de performance du temps d'exécution ils sont "programmés en dur" dans le logiciel.

3.3. Paramètres fixes et variables

L'écriture du script fait appel à un certain nombre de paramètres (par exemple la période d'une moyenne mobile, le seuil d'un signal, ...). Ces paramètres peuvent être fixés dans le script ou bien variables.

Un paramètre fixe est défini par sa valeur numérique dans le script même, et par conséquent ne peut pas être modifié sans changer sa valeur dans le script.

Un paramètre externe est désigné par un libellé dans le script, sa valeur numérique étant alors fixée ultérieurement et en dehors du script, par exemple au moment de l'utilisation de l'indicateur dans un graphique. Ceci permet d'avoir un script généraliste et de ne pas devoir le modifier pour chaque nouvelle valeur du paramètre.

3.4. Indicateur

Un indicateur est une fonction numérique du temps. Le script calcule sa valeur numérique à chaque barre. Dans un graphique, un indicateur se représente sous la forme d'une courbe au niveau du graphique principal ou d'un graphique secondaire.

3.5. Signal

Un signal est une fonction booléenne du temps. Le script calcule si sa valeur booléenne est VRAI (true) ou FAUX (false) à chaque barre. Dans un graphique, un signal se représente sous la forme de flèches verticales au niveau du graphique principal, flèches positionnées à chaque barre où le signal est VRAI.

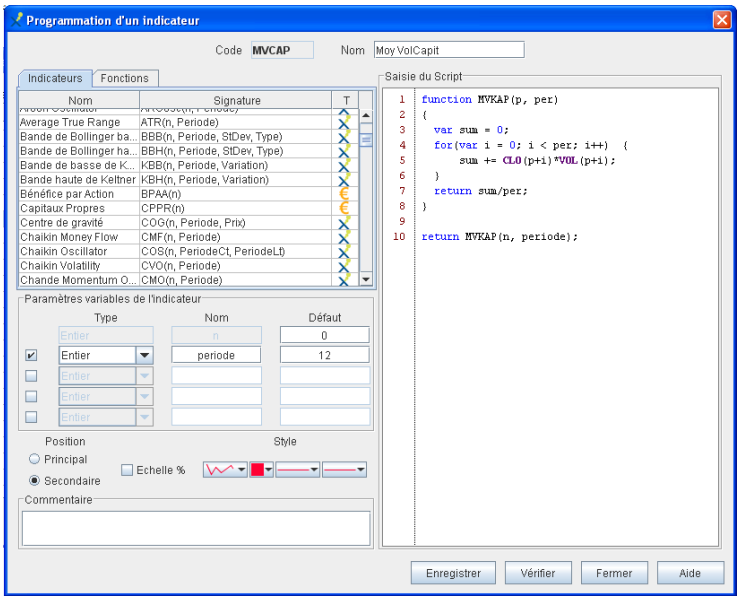
> **Note**
>
> Dans ***Axial Finance***, on distingue la notion de signal et de règle afin de simplifier le travail de programmation par l'utilisateur.
>
> Le signal est plutôt conçu pour une fonction booléenne simple, fonction nécessitant de la programmation en *JavaScript* (par exemple le cours croise la moyenne mobile).
>
> La règle est conçue pour une combinaison logique de plusieurs signaux afin de définir des fonctions booléennes plus complexes, mais sans demander de travail de programmation supplémentaire en *JavaScript* (voir dans le manuel utilisateur le chapitre relatif à la définition des règles).
>
> En fait un signal peut aussi définir une condition logique complexe et s'utiliser tel que sans passer par l'intermédiaire d'une règle. Cependant, Il est plus simple de programmer des signaux correspondant à des conditions élémentaires simples, de les avoir en bibliothèque et de les utiliser ensuite dans des règles sans avoir à refaire de la programmation en *JavaScript*.

4. Principes généreaux de la programmation

La programmation d'un indicateur et celle d'un signal sont très semblables, la différence principale étant que l'indicateur calcule une valeur numérique alors que le signal calcule une valeur booléenne VRAI ou FAUX. En outre l'indicateur est référencé de manière unique par son code mnémonique alors que le signal simplement par son nom.

4.1. Programmation d'un indicateur

Pour programmer un indicateur, au menu général ***Trading System***, cliquer sur l'option ***Indicateurs*** pour ouvrir la fenêtre d'édition des indicateurs. Dans cette fenêtre d'édition, cliquer sur le bouton **Nouveau** pour ouvrir la fenêtre de programmation ci-dessous :



La fenêtre de programmation comprend :

- En haut, un premier champ pour saisir le code de l'indicateur et un second champ pour saisir le nom de l'indicateur
- A gauche la bibliothèque des indicateurs et fonctions utilisables dans l'écriture du script, puis en dessous :
  - la zone de saisie des paramètres variables (ou externes) de l'indicateur
  - la zone de définition des caractéristiques graphiques de l'indicateur dans le cas d'une utilisation graphique
  - la zone d'enregistrement d'un commentaire
- A droite la zone d'écriture du script. En cas de besoin, la taille de la fenêtre de programmation peut être agrandie pour avoir une zone d'écriture du script plus grande.
- En bas à droite, quatre boutons :
  - **Enregistrer** pour mémoriser dans la bibliothèque l'indicateur programmé
  - **Vérifier** pour s'assurer en cours d'écriture du script que la syntaxe est correcte
  - **Fermer** pour fermer la fenêtre de programmation sans enregistrer le travail de programmation effectué
  - **Aide** pour ouvrir le manuel utilisateur directement au chapitre Trading System

Pour programmer un indicateur, il convient de suivre les étapes suivantes :

4.1.1. Définition du code

Le code de l'indicateur est essentiel, il permet de l'identifier de façon non équivoque. Par conséquent ce code doit être unique. C'est ce code qui sera utilisé dans un autre script pour désigner cet indicateur.

Les indicateurs natifs présents à l'origine dans la bibliothèque du logiciel sont généralement codés en 3 caractères alphabétiques. Par exemple :

RSI pour le code de l'indicateur "Relative Strength Index"
CLO pour le code du cours de clôture

Pour les indicateurs personnels, le code doit comprendre au moins 4 caractères alphabétiques. Si le code choisi existe déjà par ailleurs pour un autre indicateur, le logiciel demandera d'en choisir un autre.

4.1.2. Définition du nom

Le nom de l'indicateur peut être quelconque. C'est ce nom qui apparaît dans la liste des indicateurs de la bibliothèque.

4.1.3. Ecriture du script et définition des paramètres variables

Voir au paragraphe Script d'un indicateur avec ses paramètres variables ci-après

4.1.4. Définition des caractéristiques graphiques de l'indicateur

Dans ***Axial Finance***, les caractéristiques graphiques de l'indicateur ne se définissent pas dans le script du programme mais indépendamment du script dans la zone prévue à cet effet de la fenêtre de programmation.

Selon sa nature, l'indicateur programmé sera affiché dans le graphique principal ou dans un graphique secondaire et avec le style de courbe par défaut choisi à ce niveau. Si besoin, ce style pourra être modifié ultérieurement au niveau du graphique. Dans le cas d'un indicateur prévu pour un graphique secondaire, on peut sélectionner une échelle verticale en %.

4.1.5. Ajout d'un commentaire

Le cas échéant

4.1.6. Vérification de la syntaxe du script et ajout dans la bibliothèque

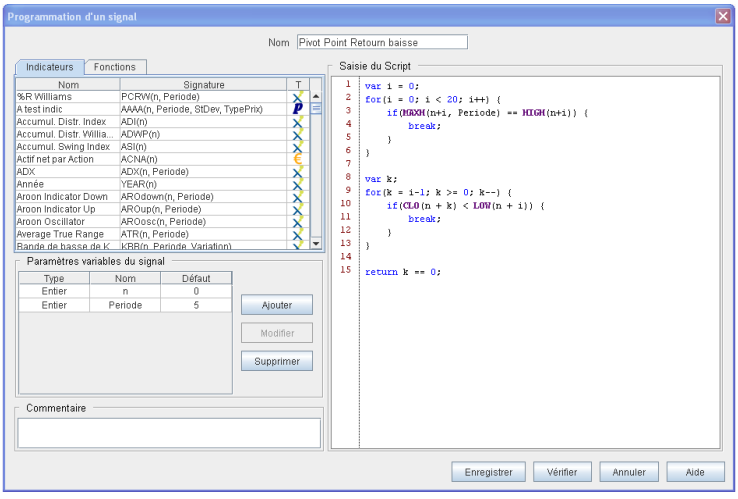
Quand l'indicateur est prêt on peut vérifier si la syntaxe est correcte en cliquant sur le bouton **Vérifier**. En cas d'erreur une fenêtre popup indique la ligne erronée du script.

Puis cliquer sur le bouton **Enregistrer** pour ajouter l'indicateur en bibliothèque et fermer la fenêtre de programmation.

Pour modifier un indicateur personnel programmé, le sélectionner dans la liste de la fenêtre d'édition et cliquer sur le bouton **Modifier** pour ouvrir la fenêtre de programmation.

4.2. Programmation d'un signal

Pour programmer un signal, au menu général ***Trading System***, cliquer sur l'option ***Signaux***
pour ouvrir la fenêtre d'édition des signaux. Dans cette fenêtre d'édition, cliquer sur le bouton **Nouveau** pour ouvrir la fenêtre de programmation ci-dessous :



La fenêtre de programmation comprend :

- En haut, un champ pour saisir le nom du signal
- A gauche la bibliothèque des indicateurs et fonctions utilisables dans l'écriture du script, puis en dessous :
  - la zone de saisie des paramètres variables (ou externes) du signal
  - la zone d'enregistrement d'un commentaire
- A droite la zone d'écriture du script. En cas de besoin, la taille de la fenêtre de programmation peut être agrandie pour avoir une zone d'écriture du script plus grande.
- En bas à droite, quatre boutons :
  - **Enregistrer** pour mémoriser dans la bibliothèque le signalr programmé
  - **Vérifier** pour s'assurer en cours d'écriture du script que la syntaxe est correcte
  - **Annuler** pour fermer la fenêtre de programmation sans enregistrer le travail de programmation effectué
  - **Aide** pour ouvrir le manuel utilisateur directement au chapitre Trading System

Pour programmer un signal, il convient de suivre les étapes suivantes :

4.2.1. Définition du nom

Le nom du signal peut être quelconque. C'est ce nom qui apparaît dans la liste des signaux de la bibliothèque.

4.2.2. Ecriture du script et définition des paramètres variables

Voir au paragraphe Script d'un signal avec ses paramètres variables ci-après

4.2.3. Ajout d'un commentaire

Le cas échéant

4.2.4. Vérification de la syntaxe du script

Quand le signal est prêt on peut vérifier si la syntaxe est correcte en cliquant sur le bouton **Vérifier**. En cas d'erreur une fenêtre popup indique la ligne erronée du script.

Puis cliquer sur le bouton **Enregistrer** pour ajouter le signal en bibliothèque et fermer la fenêtre de programmation.

Pour modifier un signal personnel programmé, le sélectionner dans la liste de la fenêtre d'édition et cliquer sur le bouton **Modifier** pour ouvrir la fenêtre de programmation.

4.2.5. Test du signal dans un graphique de cours

On peut vérifier de suite le résultat de la programmation en appliquant ce signal dans un graphique. Pour cela cliquer sur le bouton **Tester** de la fenêtre d'édition des signaux. Le signal de la bibliothèque sera appliqué sur le graphique sélectionné dans ***Ecran Graphiques***.

5. Programmation en JavaScript

5.1. Structure du langage

5.1.1. La casse (majuscules et minuscules)

En langage *JavaScript* un caractère majuscule est différent d'un caractère minuscule. Aussi, le strict respect des majuscules et minuscules est une condition indispensable au bon fonctionnement du programme.

5.1.2. Ligne d'instruction

Bien que cela ne soit pas obligatoire, il est préférable pour une bonne pratique de la programmation de terminer chaque ligne d'instruction par un point-virgule.

Exemple :

```
var dim = 10;
var condition1 = false;
var beta = 1.95;
```

La première ligne déclare la variable entière `dim` égale à 10
La seconde ligne déclare la variable boolean `condition1` égale à false
La troisième ligne déclare la variable réelle `beta` égale à 1.95

5.1.3. Les espaces et les sauts de ligne

L'insertion des espaces peut s'effectuer n'importe où dans le script comme lors d'une rédaction habituelle. Les sauts de ligne peuvent être placés partout où se trouve un espace, y compris au sein d'une instruction alors écrite sur plusieurs lignes.

5.1.4. Les commentaires

Les commentaires permettent de rendre le script plus lisible et surtout d'en faciliter ultérieurement les modifications. En général, l'insertion de commentaire se fait soit en fin de ligne, soit sur une nouvelle ligne mais en aucun cas au sein d'une instruction.

Il existe deux méthodes permettant d'intégrer des commentaires aux scripts.

La première consiste à placer un double slash `//` devant le texte.

Exemple :

```
// Mon commentaire .....
```

La seconde solution est d'encadrer le texte par un slash suivi d'une étoile `/*` et la même séquence inversée `*/`

Exemple :

```
/* Mon commentaire .....
..... sur deux lignes */
```

5.1.5. Le point

Dans les opérations mathématiques, un nombre avec décimales est séparé en France par une virgule, mais celle-ci a une signification particulière dans le langage *JavaScript*. C'est pourquoi, les nombres avec décimales doivent être séparés par un point.

Exemple :

```
var pi = 3.14159;
```

5.1.6. Les types de valeurs

Le langage *JavaScript* reconnaît plusieurs types de valeurs :

- Les nombres entiers comme `42` ou réels comme `3.14159`
- Les valeurs logiques (Booléennes) `true` (vrai) et `false` (faux)
- Les caractères comme `'a'`, `'5'`
- Les chaînes de caractères comme `"Bonjour !"`

Il n'y a aucune distinction explicite entre les nombres entiers et les nombres réels. Il n'y a également pas de type de données sous forme de dates explicites en *JavaScript*.

5.2. Les variables

Le mot-clé `var` permet de déclarer une ou plusieurs variables. Après la déclaration de la variable, il est possible de lui affecter une valeur par l'intermédiaire du signe d'égalité (=).

Si une valeur est affectée à une variable sans que cette dernière ne soit déclarée, alors *JavaScript* la déclare automatiquement. Cependant, la lecture d'une variable non déclarée provoque une erreur risquant de perturber votre programme.

D'autre-part, une variable correctement déclarée mais dont aucune valeur n'est affectée, est indéfinie (`"undefined"`). Ainsi, il est préférable de lui associer une valeur systématiquement après la déclaration.

Exemple :

```
var i, p, k;   // Déclaration de i, de p et de k
i = 1;         // Affectation de i
p = 10;        // Affectation de p
k = 3;         // Affectation de k

// Déclaration et affectation de prix
var prix = 25.3;
// Déclaration et affectation de la variable booléenne condition1
var condition1 = true;
// Déclaration et affectation de la chaîne de caractères type
var type = "Arith"
```

Nota : la chaîne de caractères se place entre guillemets.

5.3. La portée des variables

En *JavaScript*, les variables peuvent être globales ou locales. Cette notion importante s'appelle la portée des variables.

Une variable déclarée comme ci-dessus dans un script d'indicateur ou de signal avec le mot-clé `var` est considérée comme locale. C'est à dire qu'elle n'a d'existence (ou de portée) que lors de l'exécution du script à la barre courante.

> **Important**
>
> Il est fondamental de bien comprendre qu'un script d'indicateur ou de signal s'exécute à chaque barre (ou pas de calcul), chaque exécution étant indépendante de la précédente. Par principe une variable locale n'a d'existence que pour l'exécution d'un script à une barre donnée. Ainsi une variable locale ne peut pas être utilisée pour cumuler un résultat de calcul barre après barre.

Dans ***Axial Finance*** une variable globale est une variable dont l'existence perdure pendant tous les pas de calcul d'un indicateur ou d'un signal. Une variable globale ne se déclare pas avec le mot-clé var mais de la façon suivante en début du script :

```
//Assure la définition et l'affectation initiale de la variable globale total
if (typeof(total) == "undefined") {
    total = 0;
}
```

Ces lignes de code doivent se comprendre ainsi :

- Au premier pas de calcul de l'indicateur ou du signal, la variable total n'ayant pas encore été définie, son type n'est pas connu. Le script teste sa définition avec l'instruction if (typeof(total) == "undefined")
- Comme le type de la variable total est `"undefined"` le programme lui affecte la valeur initiale 0
- Ensuite, lors des pas de calcul suivants, la variable globale étant définie, l'instruction `if` ne sera plus exécutée et la variable `total` aura au début de chaque nouveau pas de calcul la valeur calculée au pas précédent.

Dans la suite de ce manuel, plusieurs exemples de script mettent en oeuvre des variables globales.

5.4. Les mots réservés

Plusieurs mots sont réservés à un emploi bien précis en *JavaScript*. C'est pourquoi, ces mots clés ne peuvent pas être utilisés pour dénommer un identificateur quelconque ce qui rendrait le programme inopérant.

Mots réservés :

`abstract`	`else`	`instanceof`	`switch`
`boolean`	`enum`	`int`	`synchronized`
`break`	`export`	`interface`	`this`
`byte`	`extends`	`long`	`throw`
`case`	`false`	`native`	`throws`
`catch`	`final`	`new`	`transient`
`char`	`finally`	`null`	`true`
`class`	`float`	`package`	`try`
`const`	`for`	`private`	`typeof`
`continue`	`function`	`protected`	`var`
`debugger`	`goto`	`public`	`void`
`default`	`if`	`return`	`volatile`
`delete`	`implements`	`short`	`while`
`do`	`import`	`static`	`with`
`double`	`in`	`super`	

5.5. Les instructions

Les instructions *JavaScript* sont contenues dans chaque ligne d'un programme. En général, elles sont fermées par un point-virgule afin que le compilateur *JavaScript* les identifie précisément.

Exemples de lignes d'instruction :

```
var sum = 0;
var per = 10 ;
sum += VOL(n + i);
return sum/per;
```

5.6. Les instructions conditionnelles

En entourant un ensemble de lignes d'instructions par des accolades, on crée un regroupement d'instructions appelé bloc d'instructions utilisable au sein, notamment, d'une commande conditionnelle telle que `if`... `else` qui les exécutera de la même manière qu'une unique instruction.

Exemple 1 :

```
if (k == 1) {
     i = i + 1;
     sum += VOL(n + i);
}
```

Exemple 2 :

```
if (k < 5) {
     i = i + 1;
     sum += VOL(n + i);
} else {
     i = i + 2;
     sum += 1.5*VOL(n + i);
}
```

L'utilisation des accolades est typique du langage *JavaScript* pour identifier les blocs de code. Une disposition régulière de ces accolades dans le script est essentielle à la bonne lisibilité du code. Plusieurs dispositions sont possibles :

Cas 1 :

```
if (k == 1)
{
     i = i + 1;
     sum += VOL(n + i);
}
```

Cas 2 :

```
if (k == 1) {
     i = i + 1;
     sum += VOL(n + i);
}
```

Cas 3 :

```
if (k == 1) { i = i + 1; sum += VOL(n + i); }
```

> **Important**
>
> Une variable peut être définie avec le mot-clé `var` dans un bloc de code. Dans ce cas cette variable est locale au bloc et n'a de portée que pour les instructions du bloc.

5.7. Les boucles

Une boucle est un bloc d'instructions qui s'exécute à plusieurs reprises jusqu'à ce qu'un état indiqué soit réuni. Le langage *JavaScript* utilise fréquemment les boucles `for` et `while`.

5.7.1. Boucle for

Exemple d'une boucle `for` qui effectue la somme des dix premiers nombres entiers :

```
var total = 0 ;
for (var k = 1; k <= 10; k = k + 1) {
    total = total + k;
}
```

L'instruction `for` contient une donnée initiale, une condition et une commande d'incrémentation ou de décrémentation, le tout séparé par des points-virgules. La boucle `for` fonctionne littéralement comme suit :

"Pour une donnée initiale (ici k = 1), jusqu'à ce que la condition soit atteinte (ici k = 10) et par pas de (ici incrémentation de k de 1 à chaque boucle) exécuter les instructions suivantes".

On peut noter que la variable `k` est identifiée et initialisée dans la boucle. Elle aurait pu être identifiée en dehors de la boucle changeant ainsi sa portée.

Remarque : La notation abrégée de `k = k+1` en `k++` est souvent utilisée en *JavaScript*.

5.7.2. Boucle while

Cette boucle `while` effectue le même calcul que la boucle `for` :

```
var total = 0 ;
var k = 1;
while (k <= 10)  {
    total = total + k;
    k++;  // Incrémentation de k
}
```

En outre, les instructions `break` et `continue` sont utilisées généralement à l'intérieur des boucles. Les instructions `break` et `continue` facilitent respectivement la sortie complète et l'arrêt pour l'itération en cours d'une boucle suite à une condition particulière nécessitant une action précise.

5.8. Les expressions et opérateurs

Les expressions permettent d'évaluer, par l'intermédiaire des opérateurs *JavaScript*, différentes valeurs ou variables littérales.

Les expressions peuvent être de trois types :

Arithmétique	exemples : `i + 1; prix = 20.50;`
Logique	exemples : `choix = true; condition = false;`
Chaîne de caractères	exemples : `adresse = "1 av Raspail"; cp = "75000"`

Le langage *JavaScript* possède un jeu complet d'opérateurs permettant de multiples combinaisons d'expressions. Ces opérateurs peuvent être mathématique et logique.

Addition	`+`
Soustraction	`-`
Multiplication	`*`
Division	`/`
Puissance	`^`
Plus grand que	`>`
Plus petit que	`<`
Plus grand que ou égal à	`>=`

Plus petit que ou égal à	`<=`
Egal à	`==`
Différent de	`!=`

ET logique	`&&`
OU logique	`||`
Inverse Logique	`!`

Important

Notez quelques différences importantes caractérisant le langage *JavaScript* :

- L'égalité entre deux expressions s'écrit avec deux caractères =, c'est à dire `==`, tandis qu'un seul signe `=` signifie l'affectation d'une valeur (ex : `periode = 5`)
- L'inégalité entre deux expressions s'écrit toujours `!=` en utilisant le caractère `!` qui caractérise toujours l'inversion.
- Le ET logique s'écrit avec les deux caractères `&&`
- Le OU logique s'écrit avec les deux caractères `||`
- Pour signifier l'inverse d'une condition, il suffit de mettre le caractère `!` devant

5.9. Les fonctions

En *JavaScript* comme dans beaucoup d'autres langages, la fonction est créée par l'instruction `function` puis son nom et entre parenthèses une liste de paramètres séparés par des virgules et entre accolades un ensemble d'instructions terminé par return qui permet de retourner le résultat de la fonction.

L'utilisation de fonctions dans un script permet de mutualiser le code pour un même calcul.

Exemple de fonction :

```
function PBAIS(n) {
    return 100*(2*(HIGH(n) - CLO(n)) - (CLO(n) -OPEN(n)))/CLO(n);
}
```

Dans la suite de ce manuel, plusieurs exemples concrets de script montreront l'intérêt de l'utilisation de fonctions.

5.10. L'instruction return

L'instruction `return` permet d'indiquer la valeur de retour d'un script ou d'une fonction.

Elle est nécessaire à la fin d'un un script pour retourner :

- la valeur numérique dans le cas d'un indicateur
- la valeur booléenne VRAI ou FAUX dans le cas d'un signal

Exemple pour un indicateur :

```
var total = 0 ;
var k = 1;
while (k <= 10)  {
    total = total + k;
```

```
    k++;        // Incrémentation de k
}
return total ;  // Le script retourne la somme des dix premiers nombres entiers
```

Exemple pour un signal :

```
var a = ADX(n+1, p) > s;    // première condition
var b = ADX(n, p) < s;      // seconde condition
return a && b;              // le script retourne le ET des deux conditions
```

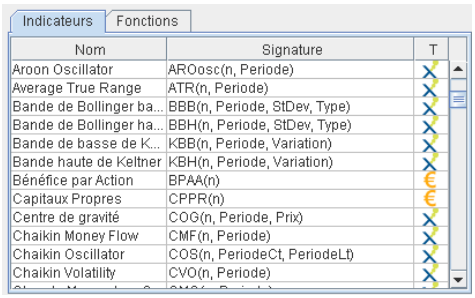
6. Script d'un indicateur avec ses paramètres variables

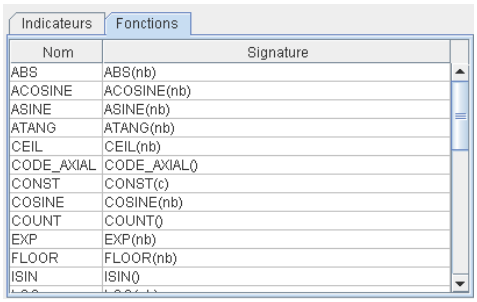
6.1. Utilisation des indicateurs et des fonctions dans le script

Lors de l'écriture d'un script on fait appel à des indicateurs (natifs ou programmés par l'utilisateur) ou des fonctions présents dans la bibliothèque du logiciel. Ces indicateurs sont référencés par un code mnémonique et un certain nombre de paramètres. Dans la fenêtre de programmation à gauche, on a accès à l'ensemble des indicateurs et fonctions utilisables pour l'écriture des scripts.

L'Annexe A au manuel de programmation donne la liste des indicateurs natifs et des fonctions disponibles.

Exemple :





Ces listes donnent le nom des indicateurs et fonctions ainsi que la signature à utiliser dans le script. La signature contient le code mnémonique et entre parenthèses les différents paramètres à prendre en compte dans le script sous la forme de paramètres variables ou fixes selon les besoins de la programmation.

Important

Dans le script il est impératif de respecter les caractères majuscules des codes mnémoniques. Pour les paramètres :

- S'il est fixé dans le script, alors son nom doit être remplacé par sa valeur effective
- S'il doit être variable, son nom peut être modifié dans le script et ce même nom reporté dans le cadre de définition des paramètres variables de la fenêtre de programmation en respectant les caractères majuscules et minuscules.

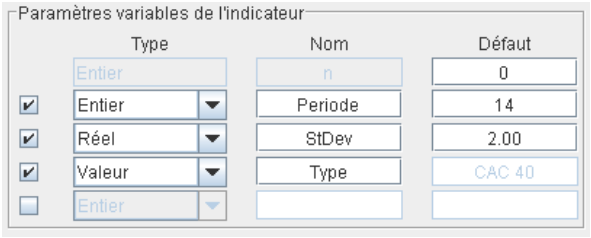
Pour aider l'utilisateur dans l'écriture du script deux dispositifs sont prévus dans ***Axial Finance*** :

- Quand un code mnémonique est écrit dans le script, celui-ci se colore immédiatement en violet et avec des caractères en gras. Ceci permet de s'assurer que le code mnémonique utilisé est bien présent dans la bibliothèque.
- Pour écrire un indicateur ou une fonction dans le script, on place le curseur de saisie du texte à l'endroit désiré puis on double clique sur sa signature dans la bibliothèque. L'ensemble du code mnémonique et des paramètres s'inscrit automatiquement dans le script.

6.2. Définition des paramètres variables de l'indicateur

En plus du paramètre n définissant la barre courante, chaque indicateur peut avoir quatre paramètres variables.

Exemple



Pour chaque paramètre variable il convient de définir :

i. Son type parmi les suivants :
   - Entier — comme la valeur d'un période
   - Réel — comme la valeur d'un seuil ou d'un nombre décimal
   - Prix — pour le cours de clôture, le plus haut, le plus bas, l'ouverture, ... de la barre
   - Moy. Mob — pour le type de moyenne mobile (Arith, Expon, Pondérée, ...)
   - Valeur — pour désigner spécifiquement une action, indice, tracker, ...
   - Mode
   - Texte
ii. Son nom en recopiant les caractères utilisés dans le script et respectant les majuscules et minuscules.
iii. Sa valeur par défaut (Nota : pour les 3 derniers types la valeur par défaut est fixé par le logiciel). La valeur par défaut sera remplacée par la valeur réelle au moment de l'utilisation effective de l'indicateur.

6.3. Rôle du paramètre temporel n

Le paramètre n sert, selon la fréquence des barres à utiliser dans le calcul de l'indicateur, à définir la date des cours à lire en mémoire.

Par exemple, considérons un graphique comportant 2547 barres en cours journaliers. Pour le calcul de l'indicateur, le logiciel itérera du premier au dernier cours et à chaque itération exécutera le script.

Lors de l'itération (ou pas de calcul) 853, la barre courante est donc la barre 853 et la valeur n = 0 du paramètre correspondra à cette barre. Si dans le script on veut désigner le cours à la barre précédente soit 852, alors on fixera le paramètre à n = 1. Et ainsi de suite, pour désigner le cours à la barre 850, on fixera le paramètre à n = 3

Ce principe de fonctionnement donne une grande souplesse pour réaliser dans le script des décalages temporels dans l'utilisation des cours.

Exemple : Calcul de la moyenne des trois derniers cours de clôture.

Le paramètre n ayant la valeur par défaut égale à 0, on écrira dans le script :

```
var moy =  (CLO(n) + CLO(n+1)  + CLO(n+2))/3;
```

où CLO est le code du cours de clôture.

Si on fixe la valeur par défaut du paramètre n = 1, alors on introduira un décalage d'une barre à gauche dans le calcul de la moyenne et le script précédent calculera la moyenne des cours aux barres 852, 851 et 850.

6.4. Codification des indicateurs les plus utilisés

Parmi les indicateurs les plus souvent utilisés dans les scripts on peut citer :

`CLO` Cours de clôture
`OPEN` Cours d'ouverture
`HIGH` Cours le plus haut
`LOW` Cours le plus bas
`VOL` Volume

`MMAR` Moyenne Mobile Arithmétique
`MMEX` Moyenne Mobile Exponentielle
`BBH` Bande de Bollinger Haute
`BBB` Bande de Bollinger Basse
`RSI` Relative Strength Index
`MACD` MACD
`MOM` Momentum
`ATR` Average True Range
`ADX` Average Directional Index
`STD` Standard Deviation
`STOD` Stochastics %D
`STOK` Stochastics %K

L'Annexe A donne la liste des indicateurs natifs fournis avec le logiciel.

6.5. Importance de l'instruction return

Comme expliqué au paragraphe L'instruction return, l'instruction `return` sert à définir la valeur de retournée par le script à chaque pas de calcul (ou la valeur retournée par une fonction).

Quand le script est simple et ne comprend par exemple qu'une seule instruction *JavaScript* permet d'omettre le mot `return`. Dans ce cas, pour calculer le pourcentage de variation par rapport au cours de la veille, le script peut se réduire à la ligne suivante :

```
100*(CLO(n)/CLO(n+1) - 1);
```

Mais pour éviter tout risque d'erreur de programmation quand le script devient plus complexe, il est recommandé de prendre l'habitude de mettre systématiquement `return`.

Dans le cas de script complexe, on peut avoir la présence de `return` plusieurs fois à l'intérieur du script, quand après l'évaluation d'une condition on veut interrompre le calcul et retourner la valeur calculée à ce niveau.

Exemple de script complexe avec plusieurs `return` qui calcule la divergence entre les cours et le RSI (cet exemple est repris en détail dans les exemples ci-après).

```
/* Fonction de calcul de la moyenne mobile Arith du RSI */
function MMARSI(n, periodeMM, periodeRSI) {
    var sum = 0;
    for (var i = 0; i < periodeMM; i++)  {
        sum += RSI(n+i, periodeRSI);
    }
    return sum/ periodeMM;
```

```
}

/* p définit la période de calcul de la pente */
var p = 5;

/* pente de la moyenne mobile de période periodeMM du RSI de période periodeRSI */
var prsi = SLOPE(MMARSI, n, p, periodeMM, periodeRSI);

/* pente de la moyenne mobile de période periodeMM des cours */
var pprix = SLOPE(MMAR, n, p, periodeMM);

/* calcul de l'écart des deux pentes */
var ecart = ABS(prsi - pprix);

/* selon le sens des pentes */
if (prsi >= 0 && pprix >= 0)  {
    return 0;
} else if (prsi < 0 && pprix < 0) {
    return 0;
} else {
    /* filtrage si écart au dessus du seuil */
    if (ecart >= seuil) {
        return ecart;
    } else {
        return 0;
    }
}
```

7. Script d'un signal avec ses paramètres variables

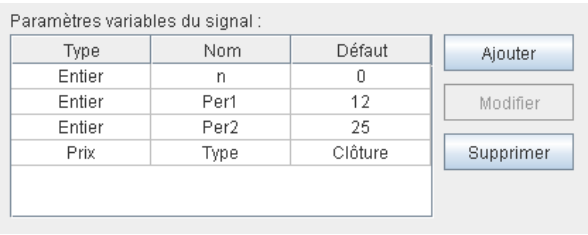
7.1. Utilisation des indicateurs et des fonctions dans le script

Voir au paragraphe

7.2. Définition des paramètres variables du signal

En plus du paramètre n définissant la barre courante, chaque signal peut avoir autant de paramètres variables que nécessaire.

Exemple



Pour chaque paramètre variable il convient de définir en cliquant sur le bouton **Ajouter** ou **Modifier** :

i. Son type parmi les suivants :

Entier	comme la valeur d'un période
Réel	comme la valeur d'un seuil ou d'un nombre décimal
Prix	pour le cours de clôture, le plus haut, le plus bas, l'ouverture, ... de la barre
Moy. Mob	pour le type de moyenne mobile (Arith, Expon, Pondérée, ...)
Valeur	pour désigner spécifiquement une action, indice, tracker, ...
Mode	

Texte

ii. Son nom en recopiant les caractères utilisés dans le script et respectant les majuscules et minuscules.

iii. Sa valeur par défaut (Nota : pour les 3 derniers types la valeur par défaut est fixé par le logiciel). La valeur par défaut sera remplacée par la valeur réelle au moment de l'utilisation effective du signal.

7.3. Rôle du paramètre temporel n

Voir au paragraphe

7.4. Codification des indicateurs les plus utilisés

Voir au paragraphe

7.5. Importance de l'instruction return

Comme expliqué dans ce paragraphe l'instruction `return` sert à définir la valeur de retournée par le script à chaque pas de calcul (ou la valeur retournée par une fonction).

Pour un signal, la valeur retournée par le script est une valeur booléenne VRAI (true) ou FAUSSE (false). A part cette différence, le reste de ce paragraphe s'applique également au signal.

8. Debugging du programme

Lors de la programmation on peut imprimer dans un fichier les résultats intermédiaires de calcul afin de vérifier si le script est conforme à l'objectif recherché. ***Axial Finance*** dispose d'un tel système de debugging.

Pour imprimer un ou plusieurs résultats intermédiaires on fait appel à la fonction `debug()` en écrivant dans le script l'instruction suivante :

```
debug(CLO(n));
```

Le résultat est imprimé dans le fichier text de nom **axial.debug.txt** du **répertoire de l'utilisateur** sur le disque dur. A chaque barre de calcul du script, une ligne est écrite dans ce fichier pour chaque instruction debug.

En outre, pour identifier plus facilement les calculs à chaque barre, on peut utiliser les fonctions `COUNT()` ou `RANK()` qui comptent la position de la barre respectivement par ordre croissant ou par ordre décroissant (0 correspondant à la dernière barre).

Exemple :

```
// ............
if(closeVeille > 0) {
    debug(RANK() + " -> closeVeille = " + closeVeille);
    if(HOUR(n) >= HOPEN(n) && HOUR(n) <= HOPEN(n) + 15) {
        debug(RANK() + " Hr : " + HOUR(n) + " -> CLO = " + CLO(n)
            + "  LIM = " + ((1 + seuilPc/100)*closeVeille));
        return CLO(n) >= (1 + seuilPc/100)*closeVeille;
    }
}
// ............
```

Chapitre 2. Exemples de scripts d'indicateurs et signaux

Ces exemples concrets sont donnés par ordre de difficulté croissante.

1. Indicateur Cours médian

```
var prixMedian = (HIGH(n) + LOW(n))/2;
return prixMedian;
```

Cet indicateur ne possède que le paramètre externe `n` avec comme valeur par défaut 0. Il sera affiché dans le graphique principal.

Il calcule la moyenne du cours le plus haut `HIGH` et du cours le plus bas `LOW`, soit la valeur médianne à la barre courante.

A noter que le sript peut s'écrire aussi en une seule instruction : `return (HIGH(n) + LOW(n))/2;`

2. Signal Volume supérieur à un seuil

```
return VOL(n) >= seuil;
```

Ce signal possède deux paramètres externes `n` et `seuil` avec comme valeur par défaut respectivement 0 et 10000 par exemple pour le seuil des volumes.

Il renvoie `true` quand le volume à la barre courante est supérieur ou égal au seuil.

3. Signal Variation Cours supérieure à %

```
var ecart = 100*(CLO(n) - CLO(n+1))/CLO(n);
return ABS(ecart) >= pcVar;
```

Ce signal possède deux paramètres externes `n` et `pcVar` avec comme valeur par défaut respectivement 0 et 2 par exemple pour la variation en pourcentage.

La première instruction calcule (en positif ou négatif) le pourcentage de variation du cours entre la barre courante et la barre précédente.

La seconde instruction prend la valeur absolue de ce pourcentage de variation en utilisant la fonction `ABS` et renvoie `true` si cette valeur absolue est supérieure à `pcVar`.

4. Indicateur Elder Ray Bull Power

```
return HIGH(n) - MMEX(n, per, type);
```

Cet indicateur possède trois paramètres externes `n`, `per` et `type` avec comme valeur par défaut respectivement 0, 10 par exemple pour la période et Clôture comme `type` de prix. Il sera affiché dans un graphique secondaire.

Il calcule l'écart à la barre courante entre le plus haut et la moyenne mobile exponentielle de période `per`. Plus cet écart est positif, plus la pression à la hausse est forte.

5. Indicateur Elder Ray Bear Power

```
return LOW(n) - MMEX(n, per, type);
```

Cet indicateur possède trois paramètres externes `n`, `per` et `type` avec comme valeur par défaut respectivement 0, 10 par exemple pour la période et `Clôture` comme >type> de prix. Il sera affiché dans un graphique secondaire.

Il calcule l'écart à la barre courante entre le plus bas et la moyenne mobile exponentielle de période `per`. Plus cet écart est négatif, plus la pression à la baisse est forte.

6. Indicateur Ecart Bandes de Bollinger

```
return BBH(n, per, stDev, type)/BBB(n, per, stDev, type);
```

Cet indicateur possède quatre paramètres externes `n`, `per`, `stDev` et `type` avec comme valeur par défaut respectivement 0, 14 par exemple pour la période des bandes de bollinger, 2.0 pour l'écart type des bandes de bollinger, et Clôture comme `type` de prix de ces bandes. Il sera affiché dans un graphique secondaire.

Cet indicateur calcule le rapport entre la bande haute et la bande basse de Bollinger. Il est donc toujours supérieur à 1, et augmente quand l'écart s'accroit.

7. Indicateur Moyenne Mobile des Volumes

```
var sum = 0;
for (var i = 0; i < perMM; i++)  {
    sum += VOL(n + i);
}
return sum/perMM;
```

Cet indicateur possède deux paramètres externes `n` et `perMM` avec comme valeur par défaut respectivement 0 et 12 par exemple. Il sera affiché dans un graphique secondaire.

La première instruction `var sum = 0;` définit et assigne à zéro la variable `sum` qui sert à cumuler les volumes.

Ensuite la boucle `for` itère de i = 0 à i = perMM - 1 pour cumuler les volumes des cours de la barre courante (i = 0) à la barre de début de la période demandée (soit perMM - 1 fois avant la barre courante).

La variable d'itération `i` est définit dans la boucle `for` et donc n'a de portée que dans cette boucle.

A noter que l'instruction `sum += VOL(n + i);` est l'écriture abrégée de `sum = sum + VOL(n + i);`

Quand l'itération est terminée, l'instruction `return` renvoie le résultat de la division du volume cumulé dans `sum` par la période égale à `perMM` pour obtenir la moyenne des volumes.

8. Indicateur Moyenne Mobile des Volumes Capitalisés

```
var sum = 0;
for (var i = 0; i < perMM; i++)  {
    sum += CLO(n + i)*VOL(n + i);
}
return sum/perMM;
```

Cet indicateur possède deux paramètres externes `n` et `perMM` avec comme valeur par défaut respectivement 0 et 12 par exemple. Il sera affiché dans un graphique secondaire.

Il est semblable au précédent mais calcule et cumule les volumes captialisés à chaque barre de la période `perMM`.

9. Indicateur Moyenne Mobile du RSI

```
var sum = 0;
for (var i = 0; i < perMM; i++) {
    sum += RSI(n+i, perRSI);
}
return sum/perMM
```

Cet indicateur possède trois paramètres externes `n`, `perMM` pour la période de la moyenne mobile et `perRSI` pour celle du RSI. Il sera affiché dans un graphique secondaire.

Il est semblable aux deux indicateurs précédents mais calcule et cumule dans la variable `sum` la valeur du `RSI` à chaque barre de la période `perMM`.

10. Signal Croisement à la hausse (ou à la baisse) des moyennes mobiles

Ce signal existe en bibliothèque du logiciel comme signal natif. Il est donné ici comme exemple pour détecter le croisement de deux moyennes mobiles, exemple facilement transposable à d'autres indicateurs en utilisant le code correspondant de l'indicateur.

```
var a = MMAR(n+1, perMM1, type) - MMAR(n+1, perMM2, type);
var b = MMAR(n, perMM1, type) - MMAR(n, perMM2, type);
return a < 0 && b >= 0;
```

Ce signal possède quatre paramètres externes `n`, `perMM1` pour la période de la première moyenne mobile, `perMM2` pour la période de la seconde moyenne mobile et `type` pour le prix à prendre en compte dans le calcul des moyennes.

Par principe, pour déterminer le croisement de deux courbes, on calcule leurs écarts à la barre courante et à la barre précédente. Il y a donc croisement quand un écart est dans un sens et l'autre écart dans l'autre sens. Le croisement est à la hausse ou à la baisse selon que l'écart à la barre précédente est négatif ou positif.

> **Important**
>
> Dans ***Axial Finance*** la signature des signaux natifs de la bibliothèque relatifs aux croisements à la hausse est écrite de la manière suivante : **MMAR(n1, Periode1) ↑ MMAR(n2, Periode2)**, mais cette signature est une écriture symbolique et en correspond en aucun cas au code à utiliser dans un script.

La première instruction calcule à la barre précédant la barre courante l'écart entre la moyenne de période `perMM1` et la moyenne de période `perMM2`. La seconde instruction calcule à la barre courante l'écart entre la moyenne de période `perMM1` et la moyenne de période `perMM2`.

L'instruction `return` renvoie `true` quand le premier écart est négatif ET (`&&`) le second écart positif ou nul, c'est à dire quand la moyenne de période `perMM1` coupe à la hausse la moyenne de période `perMM2`.

Pour un croisement à la baisse, il suffit d'écrire l'instruction `return` comme suit : `return a > 0 && b <= 0;`

11. Signal ADX coupe un seuil à la hausse

```
var a = ADX(n+1, per) < s;
var b = ADX(n, per) >= s;
return a && b;
```

Ce signal possède trois paramètres externes `n`, `per` pour la période de l'ADX et `s` pour la valeur du seuil à couper.

La première instruction affecte la variable boolean `a` à `true` si l'écart à la barre précédant la barre courante entre l'indicateur ADX et le seuil est négatif.

La seconde instruction affecte la variable boolean `b` à `true` si l'écart à la barre courante entre l'indicateur ADX et le seuil est positif ou nul.

L'instruction `return` renvoie `true` quand les deux variables boolean `a` et `b` sont `true`.

12. Signal de trois hausses successives

```
var c0 = CLO(n) > CLO(n + 1);
var c1 = CLO(n + 1) > CLO(n + 2);
var c2 = CLO(n + 2) > CLO(n + 3);
return c0 && c1 && c2
```

Ce signal possède un seul paramètre externe `n` avec comme valeur par défaut 0 pour définir la barre courante

La première instruction affecte la variable boolean `c0` à `true` si la clôture la barre courante est supérieure à celle de la barre précédente.

La seconde instruction affecte la variable boolean `c1` à `true` si la clôture la barre précédente (ou en relatif -1) est supérieure à celle de la barre antérieure (ou en relatif -2).

La troisième instruction affecte la variable boolean `c2` à `true` si la clôture à la barre -2 est supérieure à celle de la barre -3.

L'instruction `return` renvoie `true` quand les trois variables boolean `c0`, `c1` et `c2` sont `true`.

13. Indicateur Maximum Plus Haut sur une période

Cet indicateur existe dans la bibliothèque du logiciel comme indicateur natif avec la signature `MAXH(n, period)`. Il est donné ici comme exemple de programmation en langage *JavaScript* pour l'utilisation de la fonction `MAX`.

```
// Initialise la variable max avec le plus haut de la barre courante
var max = HIGH(n);
// Itération sur le reste de la période
for (var i = 1; i < per; i++)  {
    // max retient le maximum du max précédent et du plus haut de la barre
    max = MAX(max, HIGH(n + i));
}

return max;
```

Cet indicateur possède deux paramètres externes `n` et `per` pour la période. Il sera affiché dans le graphique principal.

`return` renvoie la valeur maximum des plus haut sur la période `per`

14. Indicateur Minimum Plus Bas sur une période

Cet indicateur existe dans la bibliothèque du logiciel comme indicateur natif avec la signature `MINL(n, period)`. Il est donné ici comme exemple de programmation en langage *JavaScript* pour l'utilisation de la fonction `MIN`.

```
// Initialise la variable min avec le plus bas de la barre courante
var min = LOW(n);
// Itération sur le reste de la période
for (var i = 1; i < per; i++)  {
    // min retient le minimum du min précédent et du plus bas de la barre
    min = MIN(min, LOW(n + i));
}

return min;
```

Cet indicateur possède deux paramètres externes `n` et `per` pour la période. Il sera affiché dans le graphique principal.

`return` renvoie la valeur minimum des plus bas sur la période `per`

15. Signal Cours franchit le plus haut d'une période

```
/* Calcul du maximum des plus haut avec l'indicateur MAXH
   Son premier paramètre est mis à n + 1 afin de ne pas inclure la barre courante
   dans la durée de la période */
return CLO(n) > MAXH(n + 1, period);
```

Ce signal possède deux paramètres externes `n` avec comme valeur par défaut 0 et `period` pour la période.

Ce signal renvoie `true` quand le dernier cours de la barre courante devient strictement supérieur au plus haut de la période précédente.

16. Signal Cours franchit le plus bas d'une période

```
/* Calcul du minimum des plus bas avec l'indicateur MINL
   Son premier paramètre est mis à n + 1 afin de ne pas inclure la barre courante
   dans la durée de la période */
return CLO(n) < MINL(n + 1, period);
```

Ce signal possède deux paramètres externes `n` avec comme valeur par défaut 0 et `period` pour la période.

Ce signal renvoie `true` quand le dernier cours de la barre courante devient strictement inférieur au plus bas de la période précédente.

17. Indicateur On Balance Volume

Cet indicateur existe dans la bibliothèque du logiciel comme indicateur natif avec la signature `OBV(n)`. Il est donné ici comme exemple de programmation en langage *JavaScript* pour montrer l'utilisation d'une variable globale.

```
/* Assure la définition et l'affectation initiale
   de la variable globale obv */
if (typeof(obv) == "undefined") {
    obv = 0;
}

// Variation du cours par rapport à la veille
var ecart = CLO(n) - CLO(n + 1);

/* Ajoute ou soustrait le volume
   selon le sens de l'écart */
if(ecart > 0) {
```

```
    obv += VOL(n);
} else if(ecart < 0) {
    obv -= VOL(n);
}

return obv;
```

Cet indicateur possède un paramètre externe `n` pour définir la barre courante. Il sera affiché dans un graphique secondaire.

A la première utilisation du script la variable globale `obv` est définie et initialisée à 0. Aux utilisations suivantes du script pour chaque barre, cette variable est incrémentée ou décrémentée du volume de la barre courante.

18. Signal déterminant un jour de la semaine

Ce signal simple fait appel à l'indicateur de datation `DAYW`. `DAYW(n)` définit le jour de la semaine par un nombre entier avec la convention suivante :

1 pour lundi
2 pour mardi
3 pour mercredi
4 pour jeudi
5 pour vendredi
6 pour samedi
7 pour dimanche

```
/* Détermine si le jour de la séance est un vendredi */
if(DAYW(n) == 5) {
    return true;
} else {
    return false;
}
```

Ce signal possède le seul paramètre externe `n` à 0 par défaut pour la barre courante.

Ce signal renvoie `true` quand le jour à la barre courante est vendredi.

19. Signal déterminant un jour du mois

Ce signal simple fait appel à l'indicateur de datation `DAYM`. `DAYM(n)` définit le jour du par un nombre entier avec la convention suivante :

1 le premier jour du mois
2 le second jour du mois, etc ...

```
/* Détermine si le jour est le 25 du mois */
if(DAYM(n) == 25) {
    return true;
} else {
    return false;
}
```

Ce signal possède le seul paramètre externe `n` à 0 par défaut pour la barre courante.

Ce signal renvoie `true` quand le jour du mois est le 25.

Nota : Les indicateurs `WEEK(n)`, `MONTH(n)` et `YEAR(n)` renvoient respectivement le numéro de la semaine de l'année, le mois (1 pour janvier, 2 pour février, etc ...) et l'année de la barre courante.

20. Signal déterminant une durée pendant la séance

Ce signal simple fait appel à l'indicateur de datation `HOUR` applicable en cours intraday ou en temps réel. `HOUR(n)` donne l'heure en minutes à la barre courante, étant entendu que les minutes sont comptées à partir de 0 heure dans le fuseau horaire de la bourse des cours de la valeur suivie.

```
/* Détermine si l'heure de la barre courante est comprise entre
   10 heures du matin (c'est à dire 10*60 = 600 minutes)
   et midi (c'est à dire 12*60 = 720 minutes) */
if(HOUR(n) >= 600 && HOUR(n) <= 720) {
    return true;
} else {
    return false;
}
```

Ce signal possède le seul paramètre externe `n` à 0 par défaut pour la barre courante.

Ce signal renvoie `true` quand l'heure à la barre courante est comprise entre 10 hrs et midi.

21. Signal détectant une situation en ouverture de séance

Ce signal permet de détecter une situation en ouverture de séance. Dans cet exemple la situation correspond à un gap à la hausse caractérisé par un cours en ouverture supérieur de 0.5 % à la clôture de la veille. La phase d'ouverture de séance est fixée aux quinze premières minutes.

```
/* Assure la définition de la variable globale closeVeille */
if (typeof(closeVeille) == "undefined") {
    closeVeille = 0;
}

/* initialise le cours de clôture de la veille
   à chaque passage à la barre de fermeture */
if(HOUR(n) >= HCLOSE(n)) {
    closeVeille = CLO(n);
}

/* Filtre tant que closeVeille n'est pas initialisée */
if(closeVeille > 0) {
    /* Filtre la période d'ouverture de 15 minutes */
    if(HOUR(n) >= HOPEN(n) && HOUR(n) <= HOPEN(n) + 15) {
        // Teste la condition du gap à la hausse
        return CLO(n) >= (1 + seuilPc/100)*closeVeille;
    }
}

return false;
```

Ce signal possède deux paramètres externes `n` à 0 par défaut pour la barre courante et `seuilPc` pour la valeur en % du gap.

Il utilise l'indicateur `HOPEN(n)` qui donne pour la séance de la barre courante l'heure d'ouvertue en minutes par rapport à minuit. Ainsi dans la cas d'une action d'Euronext Paris, `HOPEN(n)` est égale à 9*60 = 540 minutes. Dans le cas du contrat Future FCE, `HOPEN(n)` est égale à 8*60 = 480 minutes.

De même l'indicateur `HCLOSE(n)` donne l'heure de clôture pour la séance de la barre courante.

Il utilise la variable globale `closeVeille` pour mémoriser le cours de clôture de la veille lors des itérations successives du script à chaque barre.

Ce signal renvoie `true` si pendant les quinze premières minutes suivant l'ouverture le cours devient supérieur de 0.5% à la clôture de la veille.

22. Indicateur Oscillateur RSI

En analyse technique, un oscillateur est la différence entre deux moyennes mobiles d'un indicateur calculées à des périodes différentes. L'oscillateur permet de mettre en évidence les retournements de tendance de l'indicateur caractérisés par le passage par zéro.

Cet exemple montre l'utilisation d'une fonction dans un script.

```
// Fonction calculant la moyenne mobile arithmétique du RSI
function MMARSI(n, p1, p2) {
    var sum = 0;
    for (var i = 0; i < p1; i++)  {
        sum += RSI(n + i, p2);
    }
    return sum/p1;
}
// Calcule la différence des deux moyennes mobiles de périodes perMMCt et perMMLt
var osc = MMARSI(n, perMMCt, perRSI) - MMARSI(n, perMMLt, perRSI);

return osc;
```

Cet indicateur possède quatre paramètres externes `n`, `perRSI` pour la période du RSI, `perMMCt` pour la période de la première moyenne mobile et `perMMLt` pour la période de la seconde moyenne mobile. Il sera affiché dans un graphique secondaire.

Le premier bloc de code définit la fonction de code `MMARSI` qui possède trois paramètres propres à savoir `n`, `p1` (période de la moyenne) et `p2` (période du RSI).

La variable `osc` calcule la différence des deux moyennes mobiles en appelant deux fois la fonction MMARSI et en passant successivement la première période `perMMCt` et la seconde période `perMMLt` des moyennes mobiles.

23. Indicateur Thermomètre Elder

Cet indicateur montre la programmation d'une fonction dans un script et l'utilisation de de la fonction MOVE pour le calcul d'une moyenne mobile exponentielle.

```
/* Fonction de calcul du range courant */
function THLDR(k) {
    var dh = HIGH(k) - HIGH(k + 1);
    var db = LOW(k + 1) - LOW(k);
    var t = 0;
    if(dh > 0 || db > 0) {
        t = MAX(dh, db);
    }
    return t;
}

/* range courant */
var th = THLDR(n);

/* moyenne mobile exponentielle du range courant */
var mme = MOVE(THLDR, n, pMM);
```

```
/* Calcul et renvoie du rapport entre le range courant
   et sa moyenne mobile  */
return th/mme;
```

Cet indicateur possède deux paramètres externes `n` et `pMM` pour la période. Il sera affiché dans un graphique secondaire.

Le premier bloc de code définit la fonction de code `THLDR` qui possède le paramètre propre `k` pour définir la barre du calcul. Cette fonction renvoie le range calculé sur deux barres consécutives.

La signature générale de `MOVE` qui calcule la moyenne mobile exponentielle est la suivante `MOVE(FCT, n, p, q, r, t)` où :

`FCT` Le code de la fonction sur laquelle porte le calcul de la moyenne

`n` Le paramètre fixant la barre courante

`p` Le paramètre définissant par principe la période de la moyenne exponentielle dans la liste des paramètres de `MOVE`

`q` Un second paramètre si nécessaire relatif à `FCT`

`r` Un troisième paramètre si nécessaire relatif à `FCT`

`t` Un quatrième paramètre si nécessaire relatif à `FCT`

Important

Dans cet exemple comme la fonction `THLDR` ne prend que le seul paramètre `n` pour définir la barre courante alors seulement trois paramètres sont nécessaires pour `MOVE` :

`THLDR` pour passer le paramètre `FCT` du code de la fonction

`n` pour passer le paramètre `n` définissant toujours par principe la barre courante de calcul

`pMM` pour passer le paramètre `p` définissant toujours par principe la période de la moyenne

Les trois autres paramètres possibles de `MOVE` prévus dans sa signature sont relatifs à la fonction `FCT` à savoir `q`, `r` et `t` ne sont donc pas nécessaires à la fonction `THLDR` et peuvent donc être absents.

24. Indicateur Divergence entre cours et RSI

Cet indicateur détermine la divergence entre les cours et le RSI. La divergence est basée sur la comparaison des pentes des droites de régression linéaire des cours et de la moyenne mobile du RSI. Quand les pentes des deux droites sont opposées et que l'écart est supérieur à un seuil de filtrage, l'indicateur renvoie cet écart qui plus il est grand atteste de l'importance de la divergence.

Cet indicateur montre la programmation d'une fonction dans un script et l'utilisation de la fonction `SLOPE` à deux reprises pour le calcul des pentes des droites de régression linéaire des cours et du RSI.

```
/* Fonction calculant la moyenne mobile Arith du RSI */
function MMARSI(k, perMM, perRSI) {
    var sum = 0;
    for (var i = 0; i < perMM; i++)  {
        sum += RSI(k + i, perRSI);
    }
    return sum/perMM;
}

/* Affectation de la variable interne dureePente pour fixer la durée en nombre
   de barres  (origine -> extrémité) du calcul de la pente */
```

```
var dureePente = 5;

/* Calcul de variable penteRSI égale à la pente de la moyenne mobile
   (de période periodeMM) du RSI (de période periodeRSI) */
var penteRSI = SLOPE(MMARSI, n, dureePente, periodeMM, periodeRSI);

/* Calcul de variable penteCours égale à la pente de la moyenne mobile arithmétique
   (de période periodeMM) des cours de clôture */
var penteCours = SLOPE(MMAR, n, dureePente, periodeMM, "clôture");

/* Teste si les pentes sont de sens opposé */
if (penteRSI > 0 && penteCours < 0 || penteRSI < 0 && penteCours > 0)  {
    /* Les pentes sont de sens opposé
       on filtre alors si écart en valeur absolue est au dessus du seuil */
    var ecart = ABS(penteRSI - penteCours);
    if(ecart >= seuil) {
        return ecart;
    } else {
    /* L'écart est trop faible, on renvoie 0 pour l'indicateur */
        return 0;
    }
} else {
    /* Les pentes sont de même sens, on renvoie 0 pour l'indicateur */
    return 0;
}
```

Cet indicateur possède quatre paramètres externes `n`, `periodeMM` pour la période de la moyenne mobile du RSI, `periodeRSI` pour la période du RSI et `seuil` pour le seuil de filtrage de l'écart entre les deux pentes. Il sera affiché dans un graphique secondaire.

Le premier bloc de code définit la fonction de code `MMARSI` qui prend les trois paramètres propres `k` pour définir la barre du calcul, `perMM` pour la période de la moyenne mobile et `perRSI` pour la période du RSI. Cette fonction renvoie la moyenne mobile arithmétique du RSI.

La signature générale de `SLOPE` qui calcule la pente de la droite de régression linéaire d'un indicateur ou d'une fonction est la suivante : `SLOPE(FCT, n, p, q, r, t)` où :

`FCT` Le code de l'indicateur ou de la fonction sur lequel porte le calcul de pente de la droite de régression linéaire

`n` Le paramètre fixant la barre courante

`p` Le paramètre définissant par principe la durée en barres pour le calcul de la pente (fixant l'origine et l'extrémité de la droite)

`q` Un second paramètre si nécessaire relatif à `FCT`

`r` Un troisième paramètre si nécessaire relatif à `FCT`

`t` Un quatrième paramètre si nécessaire relatif à `FCT`

Important

Dans la première utilisation de `SLOPE` pour le calcul de la pente de la moyenne mobile du RSI, la fonction `MMARSI` prend trois paramètres `k`, `perMM` et `perRSI`. Par conséquent au total cinq paramètres sont nécessaires pour `SLOPE` :

`MMARSI`	pour passer le paramètre `FCT` du code de la fonction
`n`	pour passer le paramètre `n` définissant toujours par principe la barre courante de calcul
`dureePente`	pour passer le paramètre `p` définissant toujours par principe la durée de calcul de la pente

`periodeMM`	pour passer le paramètre `q` définissant toujours par principe la période de la moyenne mobile
`periodeRSI`	pour passer le paramètre `r` définissant toujours par principe la période du RSI

Le dernier paramètre possible de `SLOPE` prévu dans sa signature relatif à la fonction `FCT` à savoir `t` n'est pas nécessaire à la fonction `MMARSI` et peut donc être absent.

Important

Dans la seconde utilisation de `SLOPE` pour le calcul de la pente de la moyenne mobile arithmétique des cours, l'indicateur `MMAR` prend trois paramètres `n`, `periode` et `type`. Par conséquent au total cinq paramètres sont nécessaires pour `SLOPE` :

`MMAR`	pour passer le paramètre `FCT` du code l'indicateur
`n`	pour passer le paramètre `n` définissant toujours par principe la barre courante de calcul
`dureePente`	pour passer le paramètre `p` définissant toujours par principe la durée de calcul de la pente
`periodeMM`	pour passer le paramètre `q` définissant toujours par principe la période de la moyenne mobile
`"clôture"`	pour passer le paramètre `r` définissant toujours par principe le type de prix pour la moyenne mobile. A noter que ce paramètre est ici interne au script et donc passé avec sa valeur en chaîne de caracrères

Le dernier paramètre possible de `SLOPE` prévu dans sa signature relatif à la fonction `FCT` à savoir `t` n'est pas nécessaire à la fonction `MMAR` et peut donc être absent.

A noter dans cet exemple que le script ne termine pas avec l'instruction `return` en dernière ligne. Ce n'est pas nécessaire car le bloc d'instructions `if` .... `else` prévoit tous les cas possibles de fin d'exécution du script.

Annexe A. Indicateurs natifs

1. Indicateurs d'analyse technique

Code	Nom	Script
ADI	Accumulation Distribution Index	ADI(n)
ADWP	Accumulation Distribution Williams	ADWP(n)
ADX	Average Directional Index	ADX(n, Periode)
AROdown	Aaron Indicator down	AROdown(n, Periode)
AROup	Aaron Indicator up	AROup(n, Periode)
AROosc	Aaron Oscillator	AROosc(n, Periode)
ASI	Accumulation Swing Index	ASI(n)
ATR	Average True Range	ATR(n, Periode)
BBB	Bande de Bollinger Basse	BBB(n, Periode, StDev, Type)
BBH	Bande de Bollinger Haute	BBH(n, Periode, StDev, Type)
CCI	Commodity Channel Index	CCI(n, Periode)
CCIT	Commodity Channel Index Theor	CCIT(n, Periode)
CLO	Cours de clôture	CLO(n)
CMF	Chaikin Money Flow	CMF(n, Periode)
CMO	Chande Momentum Oscillator	CMO(n, Periode)
COG	Centre de Gravité	COG(n, Periode, Prix)
COS	Chaikin Oscillator	COS(n, PeriodeCt, PeriodeLt)
CRL	Courbe de régression linéaire	CRL(n, Periode, Type)
CRP	Close Range Position Index	CRP(n, Periode)
CSI	Commodity Selection Index	CSI(n, Periode)
CVO	Chaikin Volatility	CVO(n, Periode)
DBDR	Résistance DBD	DBDR(n, Rang)
DBDS	Support DBD	DBDS(n, Rang)
DEMA	Double EMA	DEMA(n, Periode)
DI	Directional Indicator	DI(n, Periode)
DPO	Detrended Price Oscillator	DPO(n, Periode)
DXm	Directional Indicator Minus	DXm(n, Periode)
DXp	Directional Indicator Positive	DXp(n, Periode)
EOM	Ease Of Movement	EOM(n)
FIX	Force Index	FIX(n, Periode)
HIGH	Cours le plus haut	HIGH(n)
KAGI	KAGI (voir nota ci-après)	KAGI(n, Mode, Seuil)
KBB	Bande haute de Keltner	KBB(n, Periode, Variation)
KBH	Bande basse de Keltner	KBH(n, Periode, Variation)

Code	Nom	Script
KLO	Klinger Oscillator	KLO(n, PeriodeCt, PeriodeLt, PeriodeTr)
LOW	Cours le plus bas	LOW(n)
LVLR	Niveau Resistance	LVLR(n, Rang)
LVLS	Niveau Support	LVLS(n, Rang)
MACD	Moving Average Convergence Divergence	MACD(n, PeriodeCt, PeriodeLt)
MACDH	MACD - Trigger	MACDH(n, PeriodeCt, PeriodeLt, PeriodeTr)
MAXH	Maximum Plus Haut	MAXH(n, Periode)
MFI	Money Flow Index	MFI(n, Periode)
MIN	Mass Index	MIN(n, Periode, PeriodeEMA)
MINL	Minimum Plus Bas	MINL(n, Periode)
MMAR	Moyenne Mobile Arithmétique	MMAR(n, Periode, Type)
MMEX	Moyenne Mobile Exponentielle	MMEX(n, Periode, Type)
MMPD	Moyenne Mobile Pondérée	MMPD(n, Periode, Type)
MMSM	Moyenne Mobile Lissée	MMSM(n, Periode, Type)
MMT3T	Moyenne Mobile T3 Tilson	MMT3T(n, Periode, Ratio)
MMTR	Moyenne Mobile Triangulaire	MMTR(n, Periode, Type)
MMVA	Moyenne Mobile Volume Ajusté	MMVA(n, Periode, Type)
MMVH	Moyenne Mobile des Volumes	MMVH(n, Periode)
MMVR	Moyenne Mobile Variable	MMVR(n, Periode, Type)
MMZL	Moyenne Mobile Zéro Lag	MMZL(n, Periode, Type)
MOM	Momentum	MOM(n, Periode)
MTSI	Moy Triangulaire du TSI	MTSI(n, Periode1, Periode2, Periode3)
MZLD	MACD Zéro Lag	MZLD(n, PeriodeCt, PeriodeLt)
NVI	Negative Volume Index	NVI(n)
OBV	On Balance Volume	OBV(n)
OCV	Open-Close Volatility Index	OCV(n, Periode)
OPEN	Cours d'ouverture	OPEN(n)
ORL	Oscillateur de Régression Linéaire	ORL(n, Periode, Type)
ORP	Open-Range Position Index	ORP(n, Periode)
OSC	Oscillateur de Moyenne Mobile	OSC(n, Periode1, Periode2, Type)
PCRW	%R Williams	PCRW(n, Periode)
POSC	Price Oscillator	POSC(n, PeriodeCt, PeriodeLt)
PRF	Performance Index	PRF(n, Type)
PRL	Prix relatifs (ou Force relative)	PRL(n, Valeur)
PROC	Price Rate Of Change	PROC(n, Periode)
PRX	Cours (clôture, ouverture, plus haut, ...)	PRX(n, Type)
PRXH	Cours Heikin Ashi (clôture, ouverture, plus haut, ...)	PRXH(n, Type)

Code	Nom	Script
PVI	Positive Volume Index	PVI(n)
PVT	Price and Volume Trend	PVT(n)
QSI	Q-Stick Indicator	QSI(n, Periode)
RMI	Relative Momentum Index	RMI(n, Periode, Ecart)
RSI	Relative Strength Index	RSI(n, Periode)
RVI	Relative Volatility Index	RVI(n, Periode)
RVO	Range Volatility Index	RVO(n, Periode)
RWIhigh	Random Walk Index High	RWIhigh(n, Periode)
RWIlow	Random Walk Index Low	RWIlow(n, Periode)
SAR	Parabolique SAR	SAR(n, MaxRatio, PcAccel)
SLOPEN	Pente DRL normalisée	SLOPEN(n, Periode, Type)
SPR	Spread	SPR(n, Valeur, Ratio)
SPTR	Super Trend VOL	SPTR(n, Periode, Type, Prix, Ratio)
STD	Standard Deviation	STD(n, Periode)
STK	Stock	STK(n, Code)
STOD	Stochastics %D	STOD(n, Periode, PeriodeK, PeriodeD)
STOK	Stochastics %K	STOK(n, Periode, PeriodeK)
STSI	Stochastics RSI	STSI(n, Periode)
SVR	Schwager Volatility Ratio	SVR(n, Periode)
SWI	Swing Index	SWI(n)
TEMA	Triple EMA	TEMA(n, Periode)
TKIN	Tick Intraday	TKIN(n, Periode, Type)
TKWE	Tick Hebdomadaire	TKWE(n, Type)
TRIX	TRIX Indicator	TRIX(n, Periode)
TSI	True Strength Indicator	TSI(n, Periode1, Periode2)
TVI	Trade Volume Index	TVI(n, Mindev)
Trend	Ratio de Tendance	Trend(n, Periode, Type)
VHF	Vertical Horizontal Filter	VHF(n, Periode)
VOL	Volume en clôture	VOL(n)
VOR	Volatility Ratio	VOR(n, Periode)
VOS	Volume Oscillator	VOS(n, PeriodeCt, PeriodeLt)
VROC	Volume Rate Of Change	VROC(n, Periode)
WUO	Ultimate Oscillator	WUO(n, PeriodeCt, PeriodeMt, PeriodeLt)

2. Indicateurs de datation

DAYM	Jour du mois	DAYM(n)
DAYW	Jour de la semaine	DAYW(n)
HCLOSE	Heure de clôture (en minutes)	HCLOSE(n)

HOPEN	Heure de d'ouverture (en minutes)	HOPEN(n)
HOUR	Heure du jour (en minutes)	HOUR(n)
MONTH	Mois de l'année	MONTH(n)
WEEK	Semaine de l'année	WEEK(n)
YEAR	Année	YEAR(n)

3. Indicateurs d'Analyse Fondamentale

ACNA	Actif Net par Action	ACNA(n)
BPAA	Bénéfice par Action	BPAA(n)
CAPI	Ratio de Capitalisation	CAPI(n)
CHAF	Chiffre d'Affaires	CHAF(n)
CPPR	Capitaux Propres	CPPR(n)
DTNT	Dette Nette	DTNT(n)
EBIT	EBIT	EBIT(n)
PER	Price Earning Ratio	PER(n)
RNET	Résultat Net	RNET(n)

4. Indicateurs spécifiques à la programmation des stratégies

Ces *indicateurs* s'utilisent uniquement pour la programmation des *règles* d'ouverture et de fermeture de position d'une *stratégie*.

DAOP	Nombre de séances de bourse depuis l'ouverture de la position	DAOP(n)
DCP	Nombre de *barres* depuis la fermeture de la position	DCP(n)
DDSC	Nombre de *barres* depuis le *signal* de fermeture	DDSC(n)
DDSO	Nombre de *barres* depuis le *signal* d'ouverture	DDSO(n)
DOPN	Nombre de *barres* depuis l'ouverture de la position	DOPN(n)
POP	Prix d'ouverture de position	POP(n)
TISC	Tick au *signal* de fermeture de position	TISC(n, Type)
TISO	Tick au *signal* d'ouverture de position	TISO(n, Type)

5. Fonctions mathématiques et utilitaires à l'analyse technique

ABS	Valeur absolue du nombre	ABS(nb)
ACOSINE	Arc Cosinus du nombre (en radian)	ACOSINE(nb)
ASINE	Arc Sinus du nombre (en radian)	ASINE(nb)
ATANG	Arc Tangente du nombre (en radian)	ATANG(nb)

CEIL	Entier immédiatement supérieur du nombre	CEIL(nb)
CODE_AXIAL	Code Axial de la valeur du script exécuté	CODE_AXIAL()
CONST	Constante de valeur c	CONST(c)
COSINE	Cosinus du nombre (en radian)	COSINE(nb)
COUNT	Compte la position la barre évaluée dans le script par ordre croissant	COUNT()
EXP	Exponentiel (base e) du nombre	EXP(nb)
FLOOR	Entier immédiatement inférieur du nombre	FLOOR(nb)
ISIN	Code ISIN de la valeur du script exécuté	ISIN()
LOG	Logarithme (base e) du nombre	LOG(nb)
LOG10	Logarithme (base 10) du nombre	LOG10(nb)
MAX	Valeur maximum de nb1 et nb2	MAX(nb1, nb2)
MIN	Valeur minimum de nb1 et nb2	MIN(nb1, nb2)
MNEMO	Code mnémonique de la valeur du script exécuté	MNEMO()
MOVA	Moyenne Mobile Arithmétique d'un indicateur	MOVA(FCT, n, p, q, r, t)
MOVE	Moyenne Mobile Exponentielle d'un indicateur	MOVE(FCT, n, p, q, r, t)
MOVP	Moyenne Mobile Pondérée d'un indicateur	MOVP(FCT, n, p, q, r, t)
PERIOD	Fréquence des cours utilisés dans le script	PERIOD()
PI	Nombre PI	PI()
POW	Valeur de nb à la puissance p	POW(nb, p)
RAND	Nombre aléatoire compris entre 0 et 1	RAND()
RANK	Compte la position de la barre évaluée dans le script par ordre décroissant (0 = barre la plus récente)	RANK()
ROUND	Valeur arrondie du nombre	ROUND(nb)
SINE	Sinus du nombre (en radian)	SINE(nb)
SLOPE	Pente de la droite de régression linéaire d'un indicateur	SLOPE(FCT, n, p, q, r, t)
SQRT	Racine carrée du nombre	SQRT(nb)
TANG	Tangente du nombre (en radian)	TANG(nb)
isCloseLong	true si l'ordre est de type clôture position long	isCloseLong()
isCloseShort	true si l'ordre est de type clôture position short	isCloseShort()
isOpenLong	true si l'ordre est de type ouverture position long	isOpenLong()
isOpenShort	true si l'ordre est de type ouverture position short	isOpenShort()