



# Guide de programmation des indicateurs et signaux

## Sommaire

1	Introduction.....	3
2	Présentation .....	3
3	Principes généraux de la programmation dans Axial Finance.....	4
3.1	Editeurs de programmation .....	5
4	Définition du vocabulaire utilisé.....	6
4.1	La notion de « Barre » .....	6
4.2	Le script.....	6
4.3	Indicateur.....	6
4.4	Codification des indicateurs .....	7
4.5	Signal.....	7
4.6	Règle .....	8
5	Exemple 1 : Indicateur simple avec paramètres fixes.....	9
5.1	Opérateurs mathématique et logique.....	10
5.2	Point virgule à la fin des lignes de code .....	10
5.3	Utilisation du mot return .....	10
5.4	Paramètres fixes .....	11
6	Exemple 2 : Indicateur simple avec paramètres variables.....	12
7	Exemple 3 : Simple avec paramètres fixes .....	13
7.1	Déclaration de variables .....	13
7.2	Expressions booléennes.....	14
8	Exemple 4 : Signal simple avec paramètres variables .....	15
9	Exemple 5 : Indicateur calculant une moyenne mobile.....	16
9.1	Expressions mathématiques abrégées.....	16
9.2	Boucle itérative « for » .....	16
9.3	Boucle itérative « while ».....	17
10	Exemple 6 : Calcul du cours le plus haut sur une période .....	18
10.1	Déclaration if .....	18
11	Exemple 7 : Programmation d'un signal complexe.....	19
11.1	Ecriture d'une fonction dans un script .....	19
11.2	Utilisation d'une fonction dans un script.....	20
11.3	Lignes de commentaire dans un script .....	20
	Annexe 1 – Liste des indicateurs pré enregistrés.....	21
	Annexe 2 – Liste des Signaux pré enregistrés.....	23
	Annexe 3 – Mots réservés en Javascript.....	25

## 1 Introduction

Ce guide explique en détail et de façon pragmatique comment programmer les indicateurs et les signaux personnels utilisables dans les règles et les stratégies du logiciel **Axial Finance Expert**.

Il est basé essentiellement sur des exemples concrets et réels de programmation, exemples 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** sont indiquées en couleur bleue.

Pour faciliter la compréhension vis à vis de lecteurs qui connaîtraient déjà d'autres langages de programmation comme « **Easy Language** » utilisé avec le logiciel **TradeStation**, ce guide indique à chaque fois l'équivalence avec ce langage.

Cette équivalence sera indiquée en couleur verte.

## 2 Présentation

Le langage de programmation utilisé par **Axial Finance** sert à définir des indicateurs et signaux personnels qui seront ajoutés aux bibliothèques contenant déjà les indicateurs et signaux pré programmés fournis avec le logiciel.

Il s'agit d'un **langage universel** largement utilisé par le monde internet, à savoir le **Javascript**.

La raison de ce choix est :

- d'une part, éviter les langages propriétaires ou spécifiques dont la pérennité n'est pas assurée,
- d'autre part, bénéficier de toutes les richesses d'un langage universel popularisé par le monde internet.

En fait, compte tenu des besoins d'**Axial Finance**, **seulement une partie de ce langage est réellement utilisée** et par conséquent nécessaire à connaître.

L'objet du présent guide est de donner les connaissances suffisantes à acquérir pour pouvoir programmer aisément avec ce langage. Ces connaissances seront acquises progressivement à partir d'exemples concrets de programmation, exemples classés par ordre de difficultés croissantes.

Pour les utilisateurs qui le souhaiteraient, de nombreux sites internet fournissent une description détaillée du langage **Javascript**.

Par exemple :

<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 html) sont sans utilité pour programmer dans Axial Finance.

Nous conseillons à l'utilisateur dans un premier temps de bien assimiler les notions contenues dans le présent document qui sont suffisantes pour programmer dans Axial Finance. Eventuellement dans un second temps certains points particuliers peuvent s'approfondir en consultant ces sites.

### 3 Principes généraux de la programmation dans Axial Finance

Pour permettre à un maximum d'utilisateur de définir des règles de screening et des stratégies, la programmation dans **Axial Finance** peut s'effectuer en fait à deux niveaux :

- Un premier niveau qui ne nécessite aucune écriture de lignes de codes. A partir des bibliothèques d'indicateurs et de signaux pré enregistrés fournis avec le logiciel, l'utilisateur définit ses règles et stratégies de **manière graphique et visuelle**.

L'objet du présent manuel n'est pas d'expliquer cette définition graphique et visuelle. Pour cela, l'utilisateur doit se reporter au chapitre « **Systemes de Trading** » du manuel utilisateur.

- Un second niveau, objet du présent Guide de programmation, pour la programmation des **indicateurs et signaux « personnels »** et qui seront alors ajoutés aux bibliothèques du logiciel.

La programmation est segmentée par sous-ensembles indépendants que l'utilisateur assemble ultérieurement sans avoir besoin d'écrire d'autres lignes de code.

Cette méthodologie expliquée dans la suite procure une grande souplesse de mise en œuvre et simplifie grandement la tâche de programmation.

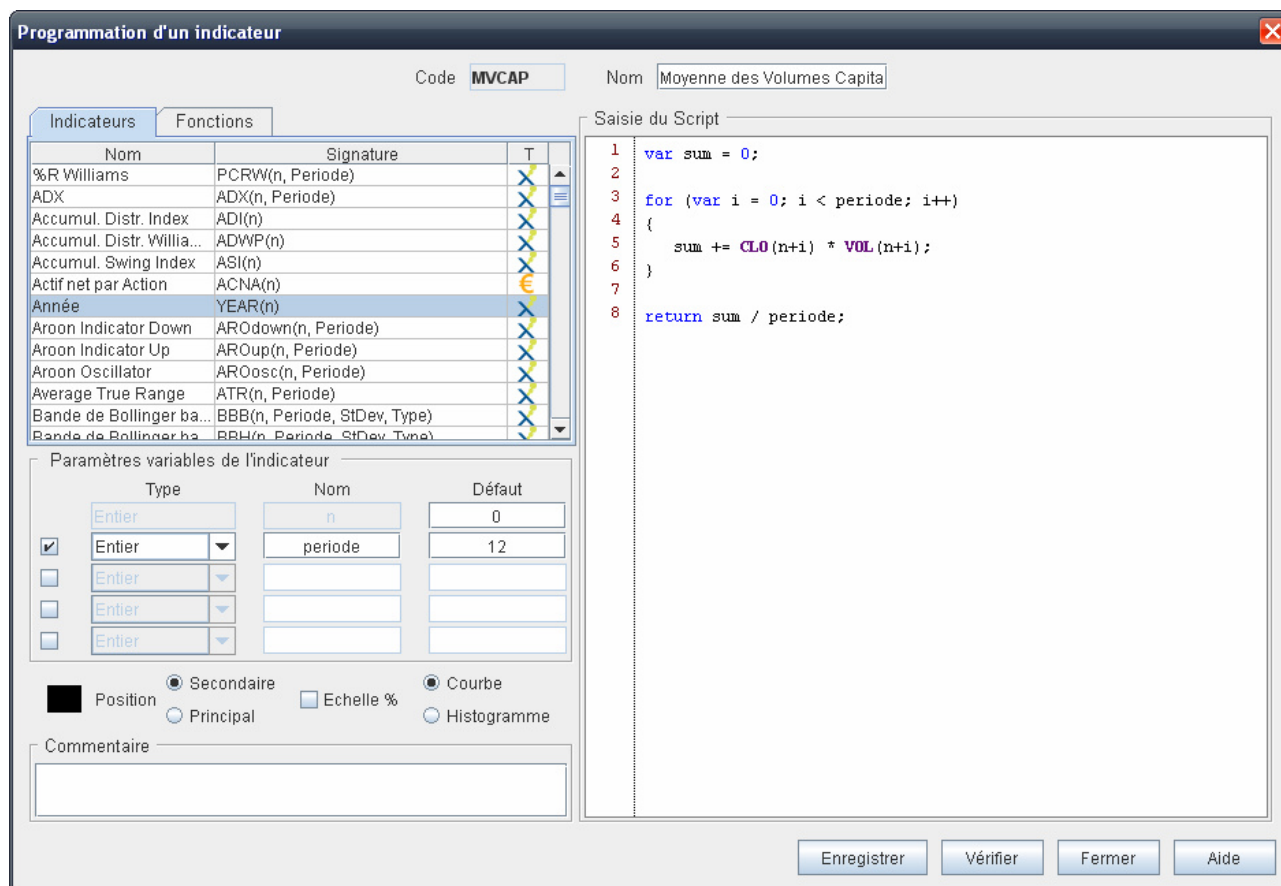
Par conséquent, **la programmation effective nécessitant l'écriture de lignes de code se réduit aux Indicateurs aux et Signaux élémentaires** (voir la définition de ces termes au paragraphe suivant).

Ensuite, la définition des règles de screening, des règles d'ouverture/fermeture de positions et des stratégies s'effectue de façon graphique et visuelle, comme avec un jeu de construction.

### 3.1 Editeurs de programmation

La programmation dans le logiciel s'effectue dans un « éditeur de programmation ».

L'image ci-dessous montre l'exemple de l'éditeur pour un indicateur. L'éditeur comprend trois parties principales :



A gauche la bibliothèque des indicateurs et fonctions disponibles. En dessous la zone de saisie des paramètres externes (ref paragraphe 6). A droite la zone d'écriture du script.

Nota : L'écriture du script est facilitée. En cliquant sur un signal dans la bibliothèque sa signature est automatiquement transférée dans le script à l'emplacement du curseur. Ceci évite de devoir frapper au clavier l'ensemble des caractères nécessaires à l'écriture de l'indicateur.

## 4 Définition du vocabulaire utilisé

Quelques définitions de vocabulaire sont nécessaires avant d'aborder le premier exemple.

### 4.1 La notion de « Barre »

Un indicateur, signal ou stratégie s'applique à une suite de cours de bourse représentés sur un graphique ou disponibles en mémoire. Chaque élément de cette suite appelé communément « **barre** » rassemble pour l'unité de temps considérée, ou encore appelée « **période** », les cours d'ouverture, de fermeture, le plus haut, le plus bas et le volume. Par exemple, on peut travailler avec des barres journalières, des barres intraday de 5 minutes, d'une heure, etc.

Le principe de l'évaluation d'un Indicateur, Signal ou Stratégie est d'effectuer le calcul pour chaque barre classée par ordre chronologique et de comparer le résultat avec celui des barres précédentes. La barre en cours d'évaluation est appelée « **barre courante** ».

Lors de la programmation des Indicateurs et Signaux, il n'y a pas à connaître a priori l'unité de temps qui sera utilisée lors de l'application car ils s'appliquent de la même façon à chaque unité. **L'unité de temps sera celle de la série de cours utilisée lors de l'application.**

### 4.2 Le script

Le **script** est tout simplement le programme (ou encore les « lignes de codes ») qui permet d'évaluer indicateur, signal et stratégies. On le saisit au clavier en écrivant dans une fenêtre prévue à cet effet.

### 4.3 Indicateur

Un indicateur (ou indicateur technique) est une **fonction** qui renvoie une **valeur numérique** à chaque unité de temps.

Un indicateur est défini par son **code mnémonique obligatoirement unique**, son nom, son **script** et par un certain nombre de **paramètres**.

Par exemple : **RSI (n, p)** pour l'indicateur « Relative Strength Index » de période p.

Vous voyez que cet indicateur contient deux paramètres mis entre parenthèses. Le premier paramètre **n** représente l'écart d'unité de temps entre la barre courante et la barre à prendre en compte pour l'évaluation.

**Dans Axial Finance, tous les indicateurs possèdent ce paramètre n ce qui donne à la programmation une très grande souplesse et des possibilités accrues.**

Par convention :

- n = 0 signifie que l'évaluation se fait à la barre courante,
- n = 1 signifie que l'évaluation se fait à la barre précédant la barre courante,
- n = 2 signifie que l'évaluation se fait deux barres avant la barre courante,
- etc ..

Par exemple, pour calculer la moyenne des trois derniers cours de clôture (par rapport à la barre courante) on écrit :

```
(CLO(0) + CLO(1) + CLO(2))/3 } (Close[0] + Close[1] + Close[2])/3
```

où **CLO** est le code du cours de clôture.

#### 4.4 Codification des indicateurs

Comme indiqué ci-dessus, chaque Indicateur est référencé par un code unique à 3 ou plus de caractères majuscules.

La bibliothèque du logiciel contient des **indicateurs pré enregistrés** permettant de satisfaire déjà de nombreuses applications.

Quand vous créez un nouvel indicateur, nous parlons d'« **indicateur personnel** ». Vous avez à choisir un code mnémotique d'au moins 4 caractères. A l'enregistrement de cet Indicateur en bibliothèque, le logiciel vérifie l'unicité du code et si ce n'est pas le cas en demande un autre.

Le nom de l'indicateur personnel n'a pas besoin d'être unique. Il est cependant fortement recommandé d'utiliser des noms évocateurs pour se souvenir ultérieurement plus facilement de sa signification précise.

Exemples de codes d'indicateurs pré enregistrés :

<b>CLO</b>	Cours de clôture de la barre	<b>Close</b>
<b>OPEN</b>	Cours d'ouverture	<b>Open</b>
<b>HIGH</b>	Cours le plus haut	<b>High</b>
<b>LOW</b>	Cours le plus bas	<b>Low</b>
<b>VOL</b>	Nombre de titres (ou de contrats)	<b>Volume</b>
<b>MMAR</b>	Moyenne Mobile Arithmétique	<b>Average</b>
<b>BBH</b>	Bande de Bollinger haute	
<b>RSI</b>	Relative Strength Index	<b>RSI</b>

La liste complète des indicateurs pré enregistrés est donnée à l'annexe 1.

#### 4.5 Signal

Un signal est une **condition qui s'évalue vraie ou fausse** à chaque barre. Un signal est défini par son **nom**, son **script** et par un certain nombre de **paramètres**.

Il peut être le résultat :

- de la comparaison de deux expressions mathématiques basées en particulier sur des Indicateurs,
- de l'existence de configurations spécifiques en chandeliers appelées également « **Patterns** »,
- du résultat du décompte d'un nombre de barres.

Le nom du signal est défini par l'utilisateur et n'a pas besoin d'être unique. Il est cependant fortement recommandé d'utiliser des noms évocateurs pour se souvenir ultérieurement plus facilement de la signification précise du signal.

Exemple de signal : cours de clôture coupe à la hausse une moyenne mobile

Le signal est vrai pour les barres courantes qui vérifient cette condition, sinon il est faux.

Le logiciel contient une bibliothèque de **signaux pré enregistrés** comme le signal ci-dessus, satisfaisant déjà de nombreux usages. Quand vous créez un nouveau signal, nous parlons de « **signal personnel** », et vous lui donnez un nom pour l'utiliser dans des règles et des stratégies.

Exemples de signaux pré enregistrés (chaque nom est explicite) :

Clôture supérieure à Ouverture  
MM Arith 1 inférieure à MM Arith 2  
RSI coupe en hausse un seuil  
ADX supérieur à un seuil  
Doji  
Tendance supérieure à un seuil

Condition1 = Close > Open

La liste des signaux pré enregistrés est donnée à l'annexe 2.

## 4.6 Règle

Une règle est une **combinaison logique de signaux**, définissant une condition complexe évaluée vraie ou fausse.

Nota : Signal et règle sont de même nature. Il est cependant utile de les distinguer car un signal est une condition générale, utilisable en de multiples occasions, et facilement disponible dans une bibliothèque. Une règle est plus spécifique, généralement construite pour un besoin précis à partir de la bibliothèque des signaux.

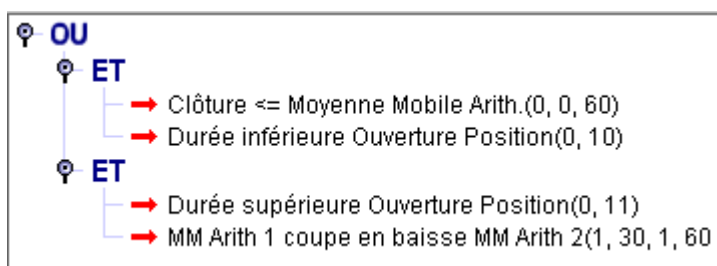
Comme pour le signal, une règle est définie par son **nom**, par la **combinaison des signaux** qui la compose et par les paramètres de ces signaux.

Les règles sont utilisées dans deux circonstances :

- Pour le « **Screening** » des marchés. Dans ce cas, la règle est seule et appliquée successivement aux valeurs du marché à une date donnée.
- Pour le « **BackTesting** » d'une stratégie sur un historique de cours. Dans ce cas, il y a deux ou quatre règles définissant les conditions d'ouverture et de fermeture des positions.

Les règles sont définies et construites par l'utilisateur, et conservées dans une bibliothèque.

Exemple de construction d'une règle :



## 5 Exemple 1 : Indicateur simple avec paramètres fixes

Commençons par un premier exemple simple d'écriture du script d'un indicateur. Appelons cet indicateur « Momentum bear » qui consiste à diviser le cours le plus bas de la barre courante par le cours le plus haut 240 barres auparavant (soit environ une année de bourse).

Dans **Axial Finance Expert** la procédure suit les étapes générales suivantes :

1. Choisir l'option « Editer Indicateurs et Signaux » du menu « Trading System »
2. Sélectionner l'onglet « Indicateurs »
3. Cliquer sur le bouton « Nouveau » puisqu'il s'agit de programmer un nouvel indicateur
4. Saisir un code et un nom pour cet indicateur
5. Ecrire le script de l'indicateur dans le cadre prévu à cet effet, et vérifier sa syntaxe en cliquant sur le bouton « Vérifier »
6. Définir si nécessaire les paramètres externes de l'indicateur (Nota : la notion de variables externes est abordée dans l'exemple 2)
7. Modifier si nécessaire les caractéristiques proposées par défaut de la représentation graphique de l'indicateur
8. Cliquer alors sur le bouton « Enregistrer » pour ajouter cet indicateur personnel dans la bibliothèque.

Ecrivons le script :

Le code du cours le plus bas (qui est lui-même un indicateur) est **LOW**

Le code du cours le plus haut (qui est lui-même un indicateur) est **HIGH**

Le script s'écrit ainsi :

```
return LOW(0)/HIGH(240);
```

```
Value1 = Low / High[240];
```

ou bien :

```
Value1 = Low / High of 240 bar ago;
```

On peut déjà noter les points suivants :

- La division est représentée par le caractère /
- Un point virgule termine la ligne de code
- Les paramètres des indicateurs **LOW** et **HIGH** sont définis par leurs valeurs numériques. Ce sont des paramètres dits « internes ».
- Le mot **return** indique que c'est le résultat qui suit qui définit la valeur de l'indicateur

## 5.1 Opérateurs mathématique et logique

Profitons de ce premier exemple pour indiquer l'écriture des opérateurs mathématique et logique en Javascript :

	Javascript	EasyLanguage
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	&&	AND
OU logique		OR
Inverse Logique	!	

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 la fixation d'une valeur (ex : value1 = 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

## 5.2 Point virgule à la fin des lignes de code

Ce premier exemple montre également que chaque ligne du script doit se terminer par un **point virgule**, comme dans beaucoup d'autres langages.

Nota : En fait la présence de ce point virgule n'est pas systématiquement obligatoire en Javascript. Cependant, pour éviter les risques d'erreur, il est conseillé d'avoir cette bonne pratique.

## 5.3 Utilisation du mot return

Dans cet exemple nous avons mis le mot `return` en début de ligne.

Par convention, comme avec les autres langages, `return` définit le résultat de l'évaluation du script qui doit être pris en compte.

IMPORTANT : le mot `return` s'écrit toujours en minuscule, il s'agit d'un mot réservé propre au langage.

Nota : Voir la liste des mots réservés à l'annexe 3.

D'une manière générale, en Javascript il convient de respecter impérativement les caractères minuscules des mots réservés, car le même caractère en minuscule ou en majuscule n'a pas la même signification.

Dans cet exemple le mot `return` aurait pu être omis car le script ne comportant qu'une seule ligne, il n'y a pas d'ambiguïté sur le résultat. De même, dans le cas général de scripts à plusieurs lignes, si la dernière ligne correspond au résultat, **Axial Finance** autorise l'omission de `return`.

Dans le cas de scripts plus complexes, quand le résultat de l'évaluation n'est pas nécessairement sur la dernière ligne, **le mot `return` est alors obligatoire.**

Pour éviter tout risque d'erreur de programmation, nous recommandons de prendre l'habitude de mettre ce mot `return` pour désigner le résultat du script.

Nota : dans certains scripts, il peut y avoir plusieurs `return` quand le résultat de l'évaluation dépend de différentes conditions.

#### 5.4 Paramètres fixes

Chaque indicateur élémentaire HIGH et LOW possède un paramètre mis entre parenthèses. Ce paramètre, appelé par convention `paramètre n`, définit la barre à laquelle l'évaluation doit être faite relativement à la barre courante comme indiqué au paragraphe 4.3

Dans cet exemple, la valeur du paramètre est fixée définitivement, selon le cas à 0 ou 240. On dit qu'il s'agit d'un paramètre fixe (ou variable interne) car non modifiable ultérieurement de l'extérieur du script, en particulier lorsque cet indicateur est inclus dans un signal ou une règle.

L'exemple qui suit va montrer l'intérêt de ne pas définir numériquement dans le script certains paramètres, c'est à dire de travailler avec des paramètres variables (ou variables externes).

## 6 Exemple 2 : Indicateur simple avec paramètres variables

Reprenons le même indicateur que dans l'exemple 1, mais cette fois-ci avec des paramètres variables.

Le script s'écrit alors :

```
return LOW(n)/HIGH(p);      Input n(0), p(240);
                             Value1 = LOW[n]/High[p];
```

Les valeurs numériques de **n** et **p** seront définies ultérieurement au lancement de l'application. **Ce sont toujours des nombres entiers positifs** pouvant varier de 0 à 2147483647 au maximum.

Rappelons que cet entier représente le nombre d'unités de temps entre la date (date en jours ou en heures selon le cas) de la barre courante et la date de la barre à laquelle l'indicateur élémentaire est calculé. Ainsi, par exemple  $n = 240$  (nombre positif) correspond à 240 barres avant la barre courante.

Ainsi, le même script pourra être utilisé de nombreuses fois en définissant à chaque fois le couple de paramètres **n** et **p** spécifique de l'application réalisée.

Cependant, il faut indiquer au moment de la programmation de l'indicateur qu'il contient deux paramètres variables. Dans Axial Finance, **cette définition s'effectue en dehors du texte du script**. Elle se fait dans le cadre de la fenêtre de programmation dénommée « **Paramètres variables de l'indicateur** ».

Chaque paramètre variable possède un nom (ici **n** ou **p**), un type (entier, nombre réel, etc.) et aussi une valeur par défaut. Dans le cas où la valeur effective du paramètre n'est pas définie au moment de l'exécution de l'application, la valeur par défaut est alors prise en compte.

## 7 Exemple 3 : Simple avec paramètres fixes

Rappelons qu'un signal est VRAI ou FAUX à une barre donnée, contrairement à un indicateur qui renvoie une valeur numérique.

Programmons le signal simple permettant de détecter la hausse de trois périodes successives de bourse. Dans **Axial Finance Expert** la procédure suit les étapes générales suivantes :

1. Choisir l'option « Editer Indicateurs et Signaux » du menu « Trading System »
2. Sélectionner l'onglet « Signaux »
3. Cliquer sur le bouton « Nouveau » puisqu'il s'agit de programmer un nouveau signal
4. Saisir un nom pour ce signal
5. Ecrire le script du signal dans le cadre prévu à cet effet, et vérifier sa syntaxe en cliquant sur le bouton « Vérifier »
6. Définir si nécessaire les paramètres externes du signal
7. Cliquer alors sur le bouton « Enregistrer » pour ajouter ce signal personnel dans la bibliothèque.

Ecrivons le script :

```
var cd1 = CLO(0) > CLO(1);
var cd2 = CLO(1) > CLO(2);
var cd3 = CLO(2) > CLO(3);

return cd1 && d2 && cd2;
```

```
Variable : cd1, cd2, cd3;
cd1 = Close[0] > Close[1];
cd2 = Close[1] > Close[2];
cd3 = Close[2] > Close[3];

If (cd1 AND cd2 AND cd3) Then
    Plot1(High, "x");
```

Nous aurions tout aussi pu écrire de manière plus compacte :

```
return CLO(0) > CLO(1) && CLO(1) > CLO(2) && CLO(2) > CLO(3);
```

La première méthode est plus facilement lisible et surtout met en valeur la manière de déclarer une variable en Javascript. Cette déclaration se fait avec le mot **var** (lettres en minuscule).

### 7.1 Déclaration de variables

Quelque soit le type de variable, la déclaration se fait toujours de la même façon :

```
var nomVariable = valeurVariable;
```

Exemples :

```
var dim = 10;
```

Déclare la variable entière **dim** égale à 10

```
var condition1 = false;
```

Déclare la variable booléenne **condition1** égale à **false**

```
var beta = 1.95;
```

Déclare la variable réelle **beta** égale à 1,95 (On utilise un point et non pas une virgule pour désigner la partie décimale d'un nombre)

```
var typeCours = "Arith";
```

Déclare une variable « chaîne de caractères » égale à Arith

Bien que facultatif en Javascript, il est recommandé de déclarer les variables utilisées dans un script.

Au moment de la déclaration, il n'est pas indispensable de fixer la valeur. On peut écrire simplement :

```
var a1;
```

Et ultérieurement dans le corps du script :

```
a1 = expression;
```

Le type de l'expression (numérique, booléenne ou chaîne de caractères) définira alors le type de la variable.

Pour alléger l'écriture, on peut déclarer plusieurs variables ensemble en les séparant par une virgule. Par exemple :

```
var a1, a3, b5, c7;
```

**IMPORTANT** : Les noms de variables sont sensibles aux caractères minuscules ou majuscules

Ainsi `typeCours` est différent de `typecours`

## 7.2 Expressions booléennes

Cet exemple met en évidence le calcul d'une expression booléenne comme :

```
var condition1 = CLO(0) > CLO(1);
```

La variable `condition1` est VRAI, c'est à dire égale à `true`, quand le cours de clôture (dont le code est `CLO`) de la barre courante (paramètre `n = 0`) est supérieur au cours de clôture de la veille (paramètre `n = 1`).

Le cumul des plusieurs conditions se fait en utilisant le ET logique qui en Javascript s'écrit avec les deux caractères accolés `&&` (voir au paragraphe 5.1).

## 8 Exemple 4 : Signal simple avec paramètres variables

Reprenons l'exemple de signal du paragraphe précédent.

Comme dans l'exemple 2, il est préférable et plus généraliste d'écrire le script du signal comme suit :

```
var d1 = CLO(n) > CLO(n+1);
var cd2 = CLO(n+1) > CLO(n+2);
var cd3 = CLO(n+2) > CLO(n+3);

return cd1 && cd2 && cd3;
```

```
Variable : cd1, cd2, cd3;
Input n(0);

cd1 = Close[n] > Close[n+1];
cd2 = Close[n+1] > Close[n+2];
cd3 = Close[n+2] > Close[n+3];

If (cd1 AND cd2 AND cd3) Then
    Plot1(High, "x" ) ;
```

Avec le paramètre `n` qui sera défini au moment de l'exécution, pouvant ainsi fixer si besoin un décalage relatif par rapport à la date de la barre courante (`n = 0`).

Cependant, il faut indiquer au moment de la programmation du signal qu'il contient un paramètre variable. Dans **Axial Finance**, cette définition s'effectue en dehors du texte du script. Elle se fait dans le cadre de la fenêtre de programmation dénommé « **Paramètres variables du signal** ».

Chaque paramètre variable possède un nom (ici `n`), un type (entier, nombre réel, etc.) et aussi une valeur par défaut. Dans le cas où la valeur effective du paramètre n'est pas définie au moment de l'exécution de l'application, la valeur par défaut est alors prise en compte.

## 9 Exemple 5 : Indicateur calculant une moyenne mobile

Prenons maintenant un exemple de script qui nécessite la mise en œuvre d'une boucle de calcul. Nous allons calculer la moyenne mobile des volumes sur les 10 dernières barres.

L'indicateur de volume est défini par le code `VOL`.

Le script est le suivant :

```
var periode = 10, sum = 0;
for (var i = 0; i < periode; i++)
{
    sum += VOL(i);
}
return sum/periode;
```

```
Variable : periode(10), sum(0);
For i = 0 To periode
Begin
    sum = sum + Volume[i];
End;
Plot1(sum/periode, "x" );
```

Cet exemple, qui ne comporte que des paramètres internes, montre la manière de programmer une boucle itérative avec l'instruction `for`, ainsi que plusieurs manières abrégées d'écrire des expressions mathématiques.

### 9.1 Expressions mathématiques abrégées

Le langage Javascript permet d'écrire de façon abrégée des expressions mathématiques fréquemment utilisées :

Par exemple, pour les plus fréquentes :

Expression	Ecriture abrégée
<code>i = i + 1</code>	<code>i++</code>
<code>i = i - 1</code>	<code>i--</code>
<code>sum = sum + a</code>	<code>sum += a</code>
<code>sum = sum - a</code>	<code>sum -= a</code>

### 9.2 Boucle itérative « for »

Pour faire varier pas à pas une variable un certain nombre de fois, ci-dessus la variable `i`, on écrit :

```
for (conditionDepart; conditionFin; variation)
{
    instructions...
}
```

où :

- `conditionDepart` définit le point de départ du calcul : `i = 0`; dans notre exemple
- `conditionFin` définit la condition de fin du calcul : le calcul s'arrête quand `i` atteint ou dépasse `periode = 10`;
- `variation` définit le pas du calcul et son sens de variation, ici `i++`; soit +1 à chaque boucle de calcul.
- `instructions...` pour signifier les lignes de codes à exécuter dans la boucle.

Notez un point important : les lignes de codes à exécuter sont encadrées par les parenthèses en forme de crochets, { et }

Cette syntaxe est caractéristique du langage Javascript : elle permet d'isoler un bloc de codes.

En fait, si le bloc de codes ne contient qu'une seule ligne, elles peuvent être omises. A partir de deux lignes elles sont impératives. Pour éviter des risques d'erreur et faciliter la lecture du code, une bonne pratique consiste donc à les mettre systématiquement.

### 9.3 Boucle itérative « while »

Pour écrire le programme précédent, nous aurions pu utiliser un autre type de boucle basé sur la déclaration `while`.

Dans ce cas, il faudrait écrire :

```
var periode = 10;
var sum = 0;
var i = 0;
while(i < periode)
{
    sum += VOL(i);
    i++;
}
return sum/periode;
```

```
Variable : i(0), periode(10), sum(0);
While(i < periode)
Begin
    sum = sum+ Volume[i];
    i = i + 1;
End;
Plot1(sum/periode, "x" );
```

## 10 Exemple 6 : Calcul du cours le plus haut sur une période

Ce nouvel exemple qui correspond en fait à l'indicateur pré programmé appelé *Maximum Plus haut* (ou « Highest(high) » en langue anglaise) va permettre d'aborder la déclaration classique `if`.

Calculons à la barre courante le cours le plus haut sur les 10 dernières barres :

<pre>var periode = 10, max = 0;  for (var i = 0; i &lt; periode; i++) {     If (max &lt; HIGH(i))         max = HIGH(i); }  return max;</pre>	<pre>Variable : periode(10), max(0);  For i = 0 To periode Begin     If (max &lt; High[i]) Then         max = High[i]; End;  Plot1(max, "x" ) ;</pre>
---	---

Vous pouvez noter dans cet exemple que nous avons mis le premier crochet de la boucle `for` en début de ligne suivante. C'est une autre manière d'écrire qui permet de mieux identifier en lecture le bloc de codes. Le second crochet aurait pu être mis à la fin de la ligne juste au dessus.

### 10.1 Déclaration `if`

Nous avons écrit :

```
if (max < HIGH(i)) max = HIGH(i);
```

Ce qui veut dire que si la condition `(max < HIGH(i))` est remplie, `max` devient égal à `HIGH(i)` ;

Nous aurions pu aussi écrire selon la syntaxe suivante :

```
if (max < HIGH(i)) { max = HIGH(i); }
```

ou encore :

```
if (max < HIGH(i))
{
    max = HIGH(i);
}
```

Cette dernière écriture entre crochets est obligatoire quand le bloc de codes comporte plusieurs lignes (comme pour les déclarations `for` ou `while`).

## 11 Exemple 7 : Programmation d'un signal complexe

Cet exemple met en œuvre l'utilisation de fonctions dans le code pour simplifier sa structuration et l'écriture du script. En outre, une fonction peut facilement se recopier et s'utiliser en tant que tel dans d'autres scripts.

En Javascript comme dans beaucoup d'autres langages, la fonction est créée par l'instruction `function` contenant 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.

Ecrivons le script du signal qui indique que la moyenne du RSI de période P est croissante trois jours de suite (ou trois barres de suite) et franchit à la hausse un certain niveau.

La fonction à programmer concerne logiquement le calcul de la moyenne du RSI de période P1 à une barre donnée car le calcul de cette moyenne doit être effectué à plusieurs reprises dans le script du signal.

### 11.1 Ecriture d'une fonction dans un script

Cette fonction s'écrit de la façon suivante :

```
function moyRSI(p, periode1)
{
    var sum = 0;
    for (var i = 0; i < periode1; i++)
    {
        sum += RSI(p + i, periode1);
    }

    return sum / periode1;
}
```

Elle est créée par l'instruction `function` contenant 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.

Les arguments sont transmis à la fonction par l'intermédiaire d'une affectation de valeur entre les arguments lors de l'appel de la fonction et les paramètres de la fonction elle-même.

La fonction est assimilable à un « bloc de codes » intégré dans le script.

Par exemple, pour calculer la moyenne du RSI de période 10 à la barre 2, il suffira alors d'écrire dans le script :

```
moyRSI (2, 10)
```

## 11.2 Utilisation d'une fonction dans un script

Ecrivons maintenant l'ensemble du script du signal contenant cette fonction et en ajoutant quelques commentaires pour faciliter la lecture :

```
/**
 * Signal du RSI de période = periodeRSI (paramètre externe),
 * en croissance trois jours de suite, coupe en hausse un certain
 * niveau = seuil (paramètre externe).
 * Le troisième paramètre externe n fixe la barre de référence
 * du signal.
 */

// Fonction calculant la moyenne du RSI de période = periodel
// à la barre p
function moyRSI(p, periodel)
{
    var sum = 0;
    for (var i = 0; i < periodel; i++)
    {
        sum += RSI(p+i, periodel);
    }
    return sum / periodel
}

// Détermination de la condition de croissance 3 jours de suite
// du RSI de periodeRSI
var rsi0 = moyRSI(n, periodeRSI);
var rsi1 = moyRSI(n + 1, periodeRSI);

var b1 = rsi0 > rsi1;
var b2 = rsi1 > moyRSI(n + 2, periodeRSI);

// Détermination de la condition de franchissement du niveau
var b3 = rsi0 > seuil && rsi1 < seuil ;

return b1 && b2 && b3 ;    // Résultat final
```

## 11.3 Lignes de commentaire dans un script

Les commentaires permettent de rendre votre code lisible et surtout d'en faciliter ultérieurement la maintenance.

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 ligne de commande**.

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

- La première consiste à placer un double slash // devant le texte comme dans l'exemple ci-dessus.
- La seconde solution est d'encadrer le texte par un slash suivi d'une étoile /\* et la même séquence inversée \*/ comme le montre aussi l'exemple ci-dessus.

## Annexe 1 – Liste des indicateurs pré enregistrés

Codes et paramètres	Nom de l'indicateur
ADI(n)	Accumulation Distribution Index
ADWP(n)	Accumulation Distribution Williams
ADX(n, Periode)	Average Directional Index
ARODown(n, Periode)	Aaron Indicator down
AROUUp(n, Periode)	Aaron Indicator up
AROosc(n, Periode)	Aaron Oscillator
ASI(n)	Accumulation Swing Index
ATR(n, Periode)	Average True Range
BBB(n, Periode, StDev, Type)	Bande de Bollinger Basse
BBH(n, Periode, StDev, Type)	Bande de Bollinger Haute
CCI(n, Periode)	Commodity Channel Index
CLO(n)	Cours de clôture
CMF(n, Periode)	Chaikin Money Flow
CMO(n, Periode)	Chande Momentum Oscillator
COS(n, PeriodeCt, PeriodeLt)	Chaikin Oscillator
CRP(n, Periode)	Close Range Position Index
CSI(n, Periode)	Commodity Selection Index
CVO(n, Periode)	Chaikin Volatility
DEMA(n, Periode)	Double EMA
DOP(n)	Nombre de cours depuis ouverture d'une position
DPO(n, Periode)	Detrended Price Oscillator
DXm(n, Periode)	Directional Indicator Minus
DXp(n, Periode)	Directional Indicator Positive
EOM(n)	Ease Of Movement
FIX(n, Periode)	Force Index
HIGH(n)	Cours le plus haut
KLO(n, PeriodeCt, PeriodeLt, PeriodeTr)	Klinger Oscillator
LOW(n)	Cours le plus bas
MACD(n, PeriodeCt, PeriodeLt, PeriodeTr)	Histogramme MACD
MASI(n, Periode, PeriodeEMA)	Mass Index
MAXH(n, Periode)	Maximum des plus hauts
MFI(n, Periode)	Money Flow Index
MINL(n, Periode)	Minimum des plus bas
MMAR(n, Periode, Type)	Moyenne Mobile Arithmétique
MMEX(n, Periode, Type)	Moyenne Mobile Exponentielle
MMPD(n, Periode, Type)	Moyenne Mobile Pondérée
MMTR(n, Periode, Type)	Moyenne Mobile Triangulaire
MMVA(n, Periode, Type)	Moyenne Mobile Volume Ajusté
MMVH(n, Periode)	Moyenne Mobile des Volumes
MMVR(n, Periode, Type)	Moyenne Mobile Variable
MOM(n, Periode)	Momentum
NVI(n)	Negative Volume Index
OBV(n)	On Balance Volume
OCV(n, Periode)	Open-Close Volatility Index
OPEN(n)	Cours d'ouverture
ORL(n, Periode, Type)	Oscillateur de Régression Linéaire
ORP(n, Periode)	Open-Range Position Index

OSC(n, Periode1, Periode2, Type)	Oscillateur de Moyenne Mobile
PCRW(n, Periode)	%R Williams
POSC(n, PeriodeCt, PeriodeLt)	Price Oscillator
PRF(n, Type)	Performance Index
PROC(n, Periode)	Price Rate Of Change
PRX(n, Type)	Cours selon type (ouverture, plus haut, etc ...)
PVI(n)	Positive Volume Index
PVT(n)	Price and Volume Trend
QSI(n, Periode)	Q-Stick Indicator
RMI(n, Periode, Ecart)	Relative Momentum Index
RSI(n, Periode)	Relative Strength Index
RVI(n, Periode)	Relative Volatility Index
RVO(n, Periode)	Range Volatility Index
RWHigh(n, Periode)	Random Walk Index High
RWLow(n, Periode)	Random Walk Index Low
SPR(n, Valeur, Ratio)	Spread
STD(n, Periode)	Standard Deviation
STOD(n, Periode, PeriodeK, PeriodeD)	Stochastics %D
STOK(n, Periode, PeriodeK)	Stochastics %K
SWI(n)	Swing Index
SWR(n, Periode)	Schwager Volatility Ratio
TEMA(n, Periode)	Triple EMA
TREND(n, Periode, Type)	Tendance Ratio
TRIX(n, Periode)	TRIX Indicator
TVI(n, Mindev)	Trade Volume Index
VHF(n, Periode)	Vertical Horizontal Filter
VOL(n)	Volume en clôture
VOR(n, Periode)	Volatility Ratio
VOS(n, PeriodeCt, PeriodeLt)	Volume Oscillator
VROC(n, Periode)	Volume Rate Of Change
WUO(n, PeriodeCt, PeriodeMt, PeriodeLt)	Ultimate Oscillator

## Annexe 2 – Liste des signaux pré enregistrés

Script générique du signal	Nom du signal
ADX(n, Periode) <= Seuil	ADX inférieur à un seuil
ADX(n, Periode) >= Seuil	ADX supérieur à un seuil
CCI(n, Periode) <= Seuil	CCI inférieur à un seuil
CCI(n, Periode) >= Seuil	CCI supérieur à un seuil
CLO(n) >= BBH(n, Periode, StDev, Type)	Clôture supérieure à Bande Bollinger H
CLO(n) <= BBB(n, Periode, StDev, Type)	Clôture inférieure à Bande Bollinger B
CLO(n1) <= MMAR(n2, periode)	Clôture inférieure à MM Arith
CLO(n1) >= MMAR(n2, Periode)	Clôture supérieure à MM Arith.
CLO(n) < OPEN(q)	Clôture inférieure à Ouverture
CLO(n) < LOW(q)	Clôture inférieure à Plus Bas
CLO(n) < HIGH(q)	Clôture inférieure à Plus Haut
CLO(n) > OPEN(q)	Clôture supérieure à Ouverture
CLO(n) > LOW(q)	Clôture supérieure à Plus Bas
CLO(n) > HIGH(q)	Clôture supérieure à Plus Haut
CLO(n) < CLO(q)	Clôtures en baisse
CLO(n) > CLO(q)	Clôtures en hausse
CLO(n) ↗ MMAR(n, Periode)	Clôture coupe en hausse sa MM Arith
CLO(n) ↘ MMAR(n, Periode)	Clôture coupe en baisse sa MM Arith.
DOP(n) = Periode	Ouverture position égale à n périodes
DOP(n) >= Periode	Ouverture position supérieure à n périodes
DOP(n) <= Periode	Ouverture position inférieure à n périodes
DXm(n, Periode) ↗ DXp(n, Periode)	DXm coupe en hausse DXp
DXp(n, Periode) ↘ DXm(n, Periode)	DXp coupe en hausse DXm
DXp(n, Periode) < DXm(n, Periode)	DXm inférieur à DXp
DXp(n, Periode) > DXm(n, Periode)	Dxp supérieur à DXm
MMAR(n1, Periode1) ↗ MMAR(n2, Periode2)	MM Arith 1 coupe en hausse MM Arith 2
MMAR(n1, Periode1) ↘ MMAR(n2, Periode2)	MM Arith 1 coupe en baisse MM Arith 2
MMAR(n1, Periode1) > MMAR(n2, Periode2)	MM Arith 1 supérieure à MM Arith 2
MMAR(n1, Periode1) < MMAR(n2, Periode2)	MM Arith 1 inférieure à MM Arith 2
MOM(n, Periode) ↗ Seuil	Momentum coupe en hausse un seuil
MOM(n, Periode) ↘ Seuil	Momentum coupe en baisse un seuil
NR(n, Periode)	Narrowest Range
Osc MACD(n, PeriodeCt, PeriodeLt) ↗	Oscillateur MACD coupe en hausse son
Trigger(PeriodeTr)	Trigger
Osc MACD(n, PeriodeCt, PeriodeLt) ↘	Oscillateur MACD coupe en baisse son
Trigger(PeriodeTr)	Trigger
Osc MACD(n, PeriodeCt, PeriodeLt) <=	Oscillateur MACD inférieur à son Trigger
Trigger(PeriodeTr)	
Osc MACD(n, PeriodeCt, PeriodeLt) >=	Oscillateur MACD supérieur à son Trigger
Trigger(PeriodeTr)	
OPEN(n) < LOW(q)	Ouverture inférieure à Plus Bas
OPEN(n) < HIGH(q)	Ouverture inférieure à Plus Haut
OPEN(n) > LOW(q)	Ouverture supérieure à Plus Bas
OPEN(n) > HIGH(q)	Ouverture supérieure à Plus Haut
OPEN(n) < OPEN(q)	Ouvertures en baisse
OPEN(n) > OPEN(q)	Ouvertures en hausse
PCRW(n, Periode) <= Seuil	%R Williams inférieur à un seuil
	%R Williams supérieur à un seuil

<p>PCRW(n, Periode) &gt;= Seuil          PROC(n, Periode) ↗ axe 0          PROC(n, Periode) ↘ axe 0          PRX(n, Type) &gt;= Seuil          PRX(n, Type) &lt;= Seuil          PRX(n1, Type1) &gt;= MMAR(n2, Periode, Type2)          PRX(n1, Type1) &lt;= MMAR(n2, Periode, Type2)          RSI(n, Periode) ↗ Seuil          RSI(n, Periode) ↘ Seuil          RSI(n, Periode) &gt;= Seuil          RSI(n, Periode) &lt;= Seuil          RSI(n, Periode) &gt; RSI(p, Periode)          RSI(n, Periode) &lt; RSI(p, Periode)          STO %K(n, Periode, PeriodeK) &gt;= Seuil          STO %K(n, Periode, PeriodeK) &lt;= Seuil          STO %K(n, Periode, PeriodeK) ↗ STO %D(n, Periode, PeriodeK, PeriodeD)          STO %K(n, Periode, PeriodeK) ↘ STO %D(n, Periode, PeriodeK, PeriodeD)          TREND(n, Periode) &lt;= Seuil          TREND(n, Periode) &gt;= Seuil          VOL(n) &gt;= Ratio * MMVH(n, Periode)</p>	<p>Price ROC coupe en hausse le seuil 0          Price ROC coupe en baisse le seuil 0          Cours supérieur à un seuil          Cours inférieur à un seuil          Cours supérieur à sa MM Arith          Cours inférieur à sa MM Arith          RSI coupe en hausse un seuil          RSI coupe en baisse un seuil          RSI supérieur à un seuil          RSI inférieur à un seuil          RSI n supérieur à RSI p          RSI n inférieur à RSI p          Stochastics %K supérieure à un seuil          Stochastics %K inférieure à un seuil          Stochastics %K coupe en hausse Stochastics %D          Stochastics %K coupe en baisse Stochastics %D          Tendance inférieure à un seuil          Tendance supérieure à un seuil          Volume supérieur à son Volume Moyen</p>
<p>Bearish Chandelier(n)          Bearish Counter Attack(n)          Bearish Eng(n)          Bearish Eng Conf(n)          Bearish Harami(n)          Bearish Harami confirmé(n)          Bearish Kicker(n)          Bullish Chandelier(n)          Bullish Counter Attack(n)          Bullish Eng(n)          Bullish Eng Conf(n)          Bullish Harami(n)          Bullish Harami confirmé(n)          Bullish Kicker(n)          Ciel couvert(n)          Doji(n)          Etoile du matin(n)          Etoile du soir(n)          Ligne perçante(n)          Marteaux(n)          Marteaux inversés(n)          Pendu(n)          Three Black Crows(n)          Three White Soldiers(n)</p>	<p>Bearish chandelier          Bearish Counter Attack          Bearish Engulfing          Bearish Engulfing confirmé          Bearish Harami          Bearish Harami confirmé          Bearish Kicker          Bullish chandelier          Bullish Counter Attack          Bullish Engulfing          Bullish Engulfing confirmé          Bullish Harami          Bullish Harami confirmé          Bullish Kicker          Ciel couvert          Doji          Etoile du matin          Etoile du soir          Ligne perçante          Marteaux          Marteaux inverses          Pendu          Three Black Crows          Three White Soldiers</p>

## Annexe 3 – Mots réservés en Javascript

Les mots réservés dans la liste ci-après ne peuvent pas être utilisés comme variables Javascript, fonctions, ou noms de méthodes.

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