ADOBE® INTRODUCTION À L'ÉCRITURE DE SCRIPTS

Adobe

Adobe® Introduction à l'écriture de scripts

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, Etats-Unis.

Table des matières

1 Introduction

L'écriture de scripts est un outil puissant qui vous permet de contrôler et d'automatiser un grand nombre de fonctionnalités dans un grand nombre d'applications Adobe® Creative Suite® et vous permet de gagner tellement de temps et d'efforts que votre approche du travail s'en trouve complètement changée.

Est-ce difficile de créer des scripts ?

Les scripts sont différents des programmes. Il n'est pas nécessaire de disposer d'un diplôme en informatique ou en mathématiques pour créer des scripts pouvant automatiser un vaste éventail de tâches courantes.

Chaque élément de script correspond à un outil, une palette ou un menu d'une application Adobe. En d'autres termes, chaque élément de script correspond à des connaissances que vous possédez déjà sous Adobe. Si vous savez ce que vous souhaiteriez que vos applications Adobe fassent, il ne vous sera pas difficile d'apprendre à écrire un script.

Pourquoi utiliser un script ?

Votre travail de conception graphique est caractérisé par la créativité, mais, en pratique, la plupart des tâches n'ont rien de créatif. Il y a de fortes chances que vous passiez le plus clair de votre temps à exécuter des tâches répétitives ou similaires.

Ne serait-il pas pratique de pouvoir embaucher un assistant heureux d'exécuter les tâches ennuyeuses, de suivre vos instructions de manière prévisible et parfaitement régulière, qui soit disponible à tout moment, travaille à la vitesse de la lumière et ne vous envoie jamais de facture ?

Cet assistant, ce peut être le script. Il ne faut pas longtemps pour apprendre à générer des scripts permettant d'exécuter les tâches répétitives qui vous font perdre votre temps. Mais s'il est facile de s'y mettre, les langages de script modernes sont assez sophistiqués pour automatiser des tâches très compliquées. Au fur et à mesure que votre expérience s'accroît, vous pourrez passer à des scripts plus complexes exécutables de nuit pendant votre sommeil.

Comment savoir quand utiliser un script ?

Pensez à votre travail : ne comporte-t-il pas une tâche répétitive qui vous ennuie au plus haut point ? Si c'est le cas, voilà déjà une bonne raison d'écrire un script. Il vous suffira ensuite de déterminer :

- quelles sont les étapes nécessaires à l'exécution de cette tâche et?
- quelles sont les conditions dans lesquelles vous devez effectuer cette tâche?

Une fois que vous avez compris le processus d'exécution manuelle de cette tâche, vous êtes prêt à en écrire le script.

Qu'en est-il des actions ou des macros ?

Si vous avez utilisé des actions ou rédigé des macros, vous pouvez vous faire une idée de l'efficacité des scripts. Mais l'écriture de scripts va au-delà de l'automatisation d'actions ou de macros ; elle permet de manipuler des documents et des applications multiples en un seul script. Par exemple, vous pouvez créer un script qui manipule une image sous Adobe Photoshop® et ordonne ensuite à Adobe InDesign® d'incorporer cette image.

En outre, votre script peut très intelligemment retrouver des informations et y répondre en conséquence. Par exemple, si vous avez un document qui contient des photos de différentes tailles. vous pouvez créer un script qui sélectionne la taille de chaque photo et crée une bordure colorée différente selon chaque taille. Ainsi, les formats icônes peuvent avoir une bordure bleue, les petites illustrations une bordure verte et les photos d'une demi-page une bordure argentée.

Si vous aimez utiliser des actions, n'oubliez pas que votre script peut exécuter ces dernières dans l'application.

L'écriture de scripts, c'est quoi au juste ?

Un script est une série d'instructions qui ordonnent à une application d'exécuter un ensemble de tâches.

Le truc consiste à écrire ces instructions dans un langage reconnu par l'application. Les applications Adobe pour lesquelles vous pouvez écrire des scripts prennent en charge plusieurs langages.

Si vous travaillez sous Mac OS®, vous pouvez utiliser :

- AppleScript
- JavaScript

Si vous travaillez sous Windows®, vous pouvez utiliser :

- VBScript (Visual Basic et VBA fonctionnent également)
- JavaScript

Les brèves descriptions ci-dessous peuvent vous aider à déterminer le langage qui vous conviendra le mieux.

AppleScript

AppleScript est un langage de script « en langue courante » développé par Apple. C'est l'un des langages de script les plus simples à utiliser.

Pour écrire des scripts AppleScript, vous pouvez utiliser l'application Editeur de scripts d'Apple, qui, dans une installation par défaut Mac OS, se trouve dans :

```
disque système:Applications:AppleScript:Editeur de scripts
```

Pour plus d'informations sur l'utilisation de l'application Editeur de scripts, reportez-vous à l'aide correspondante.

VBScript :

VBScript est un sous-ensemble du langage de programmation Visual Basic développé par Microsoft. VBScript communique avec les applications hébergées à l'aide d'ActiveX Scripting. Si VBScript est la version du langage Visual Basic officiellement prise en charge par CS6, vous pouvez également écrire des scripts en VBA et Visual Basic lui-même.

Vous pouvez trouver plusieurs bons éditeurs VBScript sur Internet. Si vous avez une application Microsoft Office, vous pouvez également utiliser l'éditeur Visual Basic intégré en sélectionnant **Outils > Macro > Visual Basic Editor**.

JavaScript

JavaScript est un langage de script très répandu développé à l'origine pour rendre les pages Web interactives. Tout comme AppleScript, JavaScript est facile à apprendre.

REMARQUE : Adobe a développé une version améliorée de JavaScript appelée ExtendScript, qui vous permet de bénéficier des avantages de certains outils et fonctionnalités de script d'Adobe. Si vous êtes débutant, la différence entre ces deux langages ne devrait pas vous affecter. Prenez toutefois l'habitude de donner une extension `.jsx` à vos scripts JavaScript, plutôt que l'extension habituelle `.js`.

JavaScript possède quelques petits avantages sur AppleScript et Visual Basic :

- Vos scripts peuvent être utilisés soit sous Windows, soit sous Mac OS. Si vous êtes amené à partager ou à utiliser vos scripts dans les deux plates-formes, apprenez alors à utiliser JavaScript.
- Sous Adobe Illustrator® et InDesign, vous pouvez accéder aux scripts dans n'importe quel langage pris en charge à partir de l'application. Néanmoins, sous Photoshop, vous ne pouvez accéder qu'aux fichiers `.jsx` à partir de l'application. Vous devez utiliser les scripts AppleScript ou Visual Basic en dehors de l'application. Cela ne constitue pas un inconvénient majeur, mais il vous faudra quelques clics de plus pour exécuter vos scripts.
- Vous pouvez configurer les scripts `.jsx` pour qu'ils s'exécutent automatiquement lorsque vous ouvrez l'application, en les plaçant dans le dossier Startup Scripts de l'application. Pour plus d'informations sur les dossiers Startup Scripts, reportez-vous au guide d'écriture de scripts de votre application.

Pour écrire des scripts en JavaScript, vous pouvez utiliser n'importe quel éditeur de texte, ou encore l'ESTK (ExtendScript Tool Kit) fourni avec vos applications Adobe. L'ESTK possède un large éventail de fonctionnalités qui rendent son utilisation plus facile qu'un éditeur de texte, notamment grâce à un correcteur de syntaxe intégré qui identifie les problèmes liés à votre script et suggère des moyens de les résoudre, et grâce à la possibilité d'exécuter vos scripts directement à partir de l'ESTK sans sauvegarder le fichier. Cette deuxième fonctionnalité vous permet de gagner beaucoup de temps, surtout au début s'il vous faut tester et modifier un script plus d'une fois avant que celui-ci ne soit prêt à fonctionner.

Dans une installation Adobe par défaut, l'ESTK se trouve à l'emplacement suivant :

Mac OS : *`disque système`*`:Applications:Utilities:Adobe Utilities - CS6:ExtendScript Toolkit CS6`

Windows : *`lecteur`*`:/Program Files/Adobe/Adobe Utilities - CS6/ExtendScript Toolkit CS6`

Pour plus d'informations, reportez-vous au *Guide JavaScript Tools*.

Par où dois-je commencer ?

Il est temps d'écrire votre premier script.

REMARQUE : Si vous rencontrez des problèmes pour exécuter votre script, reportez-vous au Chapitre 5, « Résolution des problèmes ».

AS

1. Ouvrez l'Editeur de scripts et tapez (en changeant le nom de l'application Adobe entre guillemets) :

```
tell application "Adobe Photoshop CS6"
    make document
end tell
```

2. Cliquez sur **Exécuter**.

JS

1. Ouvrez l'ESTK et sélectionnez une application dans la liste déroulante située en haut à gauche de la fenêtre du document.
2. Dans la palette Console de JavaScript, tapez :

```
app.documents.add()
```

3. Effectuez l'une des opérations suivantes :

- Cliquez sur l'icône Exécuter dans la barre d'outils située en haut de la fenêtre du document.
- Appuyez sur la touche **F5**.
- Sélectionnez **Debug (Débogage) -> Run (Exécuter)**.

VBS

1. Dans un éditeur de texte, tapez (en changeant le nom de l'application Adobe entre guillemets de la deuxième ligne) :

```
Set appRef = CreateObject("Photoshop.Application")
appRef.Documents.Add()
```

2. Sauvegardez le fichier au format texte avec une extension `.vbs` (par exemple, `creer_doc.vbs`).
3. Double-cliquez sur le fichier dans Windows Explorer.

2 Bases de l'écriture de scripts

Ce chapitre présente les concepts de base de l'écriture de scripts sous Windows et sous Mac OS. Pour des instructions spécifiques à un produit, reportez-vous au guide d'écriture de scripts de votre application Adobe.

Modules de l'écriture de scripts

Votre premier script, qui a créé un nouveau document, a été construit comme une phrase en français, avec un nom (`document`) et un verbe (`make` en langage AS, `add()` en langage JS et `Add` en langage VBS). Dans l'écriture de scripts, un nom est appelé un *objet* et un verbe une *commande* (en AS) ou une *méthode* (en JS et VBS).

Exactement comme vous pouvez modifier un nom en utilisant un adjectif, vous pouvez modifier un objet de script en utilisant des *propriétés*. Pour modifier une commande ou une méthode, vous utilisez des *paramètres*.

Utilisation des objets, des propriétés, des méthodes et des commandes

Lorsque vous utilisez une application Adobe, vous ouvrez un fichier ou un document, puis, au sein du document, vous créez ou manipulez des calques, du texte, des blocs, des couches, des lignes de graphique, des couleurs et d'autres éléments créatifs. Ces éléments sont des objets.

Pour créer une instruction de script, vous créez un objet ou faites référence à un objet existant, puis vous procédez comme suit :

- Définissez des valeurs pour les propriétés de l'objet. Par exemple, vous pouvez indiquer le nom, la hauteur ou la largeur d'un document. Vous pouvez spécifier le nom, la couleur ou l'opacité d'un calque.
- Spécifiez des commandes ou des méthodes pour indiquer au script ce qu'il doit faire à vos objets. Par exemple, vous pouvez ouvrir, fermer, enregistrer et imprimer un document. Vous pouvez fusionner, déplacer ou pixelliser un calque.

Gardez à l'esprit lorsque vous rédigez un script que vous ne pouvez utiliser que les propriétés ou méthodes/commandes autorisées pour l'objet. Comment savoir quelles propriétés et méthodes sont autorisées pour quel objet ? La plupart du temps, c'est logique. En général, si vous pouvez indiquer quelque chose dans votre application Adobe, vous pouvez l'indiquer dans un script.

Néanmoins, Adobe l'explique également en détail dans les ressources de l'écriture de scripts qui contiennent les informations dont vous avez besoin pour créer, définir et manipuler les objets de script. Pour plus d'informations sur l'emplacement et l'utilisation de ces ressources, reportez-vous au Chapitre 3, « Recherche de propriétés et de méthodes d'objets ».

Utilisation des objets

Le principal concept à comprendre pour utiliser des objets dans des scripts est de savoir comment faire référence à un objet. Comment faire savoir à l'application quel objet vous voulez que votre script modifie ? Dans l'interface utilisateur de l'application, vous pouvez simplement sélectionner l'objet en cliquant dessus. Dans un script, il y a un peu plus à faire.

Concepts DOM

Les langages de script utilisent un *Document Object Model* (DOM) pour organiser les objets de façon à ce qu'ils soient faciles à identifier. Le principe sous-jacent un DOM est la *hiérarchie de confinement*. En d'autres termes, les objets de niveau supérieur *contiennent* les objets du niveau suivant, qui contiennent le niveau suivant d'objets, etc.

Par exemple, l'objet de niveau supérieur de tout DOM d'une application Adobe est l'objet application. Ensuite vient l'objet document qui contient tous les autres objets, comme les calques, les couches, les pages, les blocs de texte, etc. Ces objets peuvent contenir des objets que le document ne peut pas contenir directement. Par exemple, sous InDesign ou Illustrator, un bloc de texte peut contenir des mots. Un document ne peut pas contenir de mots sans bloc de texte. De même, sous Photoshop, un document peut contenir un calque et un calque peut contenir un bloc de texte mais un document ne peut pas contenir de bloc de texte si le document ne contient pas de calque.

REMARQUE : Un objet contenant un objet est également appelé objet *parent*.

Dans votre premier script, vous avez tout d'abord nommé l'objet application (ou vous l'avez sélectionné dans l'ESTK), puis vous avez créé le document dans cette application. Si, comme étape suivante, vous vouliez créer un calque, votre script aurait eu besoin d'identifier le document dans lequel vous voulez créer le calque. Si votre script n'indique pas à l'application exactement où créer un objet, votre script échoue.

REMARQUE : Pour consulter un tableau du DOM pour une application spécifique, reportez-vous au guide d'écriture de scripts de l'application.

Donc, en utilisant votre principe DOM, comment ajoutez-vous un calque à un document ? (Pour modifier ce script pour Photoshop, notez qu'un calque est appelé `art layer` en AS et les calques sont appelés `artLayers` en JS ou `ArtLayers` en VBS).

AS

```
tell application "Adobe Illustrator CS6"
    make document
    make layer in document
end tell
```

JS

```
app.documents.layers.add()
```

VBS

```
Set appRef = CreateObject("Illustrator.Application")
docRef.Documents.Add
appRef.Documents.Layers.Add
```

Si vous tentez d'exécuter ces scripts, vous obtiendrez une erreur car l'application ne sait pas quel document choisir. Bien sûr, vous n'avez qu'un seul document d'ouvert mais cela ne sera pas toujours le cas.
En conséquence, les langages de script sont soumis à des conditions strictes spécifiant que tous les objets soient explicitement identifiés dans chaque instruction de script.

Ce guide présente trois moyens de faire référence aux objets :

- Variables
- Numéros d'ensemble ou d'élément
- Objet « actuel » ou propriété de l'objet « actif »

Variables

Une *variable* est un élément que vous créez pour contenir des données dans votre script. Les données, appelées *valeur* de la variable, peuvent être un objet de votre script ou une propriété qui décrit un objet. Vous pouvez également considérer une variable comme un surnom que vous donnez à un objet ou à d'autres données.

L'utilisation d'une variable pour contenir un objet facilite la référence à cet objet. La plupart des rédacteurs de script créent une variable pour chaque objet de leur script.

Les scripts suivants créent un document, comme vous l'avez fait dans votre premier script. Néanmoins, cette version du script crée une variable nommée `myDoc` pour contenir le document. Etudiez ces scripts, puis comparez-les à votre premier script. (Reportez-vous à « Par où dois-je commencer ? », page 8.)

AS Pour créer une variable en langage AS, utilisez la commande `set` suivie par le nom de la variable. Pour affecter une valeur de données à la variable, utilisez `to` suivi de la valeur.

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
end tell
```

JS Pour créer une variable en langage JS, utilisez la commande `var` suivie par le nom de la variable. Pour affecter une valeur de données, utilisez le symbole égal (=) suivi de la valeur. Les espaces n'ont aucune importance de quelque côté que ce soit du symbole égal.

```
var myDoc = app.documents.add()
```

VBS Pour créer une variable en langage VBS, utilisez la commande `Set` suivie du nom de la variable. Pour affecter une valeur de données, utilisez le symbole égal (=) suivi de la valeur. Les espaces n'ont aucune importance de quelque côté que ce soit du symbole égal.

```
Set appRef = CreateObject("Illustrator.Application")
Set docRef = appRef.Documents.Add
```

Les références d'objet facilitent la vie

Maintenant que vous connaissez une méthode pour faire référence à l'objet `document` créé dans le script, il est facile d'ajouter le calque. (Pour modifier ce script pour Photoshop, notez qu'un calque est appelé `art layer` en AS et les calques sont appelés `artLayers` en JS ou `ArtLayers` en VBS).

AS

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    make layer in myDoc
end tell
```

Mieux : vous pouvez créer une autre variable pour contenir le calque. Cela nous permettrait de faire facilement référence au calque si nous souhaitions définir ses propriétés ou ajouter un objet au calque.

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc
end tell
```

JS

```
var myDoc = app.documents.add()
    myDoc.layers.add()
```

Le même script à nouveau, cette fois-ci en créant une variable pour contenir le calque.

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
```

VBS

```
Set appRef = CreateObject("Illustrator.Application")
Set docRef = appRef.Documents.Add
    docRef.Layers.Add
```

Le même script à nouveau, cette fois-ci en créant une variable pour contenir le calque.

```
Set appRef = CreateObject("Photoshop.Application")
Set docRef = appRef.Documents.Add
Set layerRef = docRef.Layers.Add
```

Les variables offrent un raccourci convivial

Les variables qui contiennent des objets contiennent également l'ensemble de la hiérarchie de confinement qui identifie l'objet. Par exemple, pour faire référence à `myLayer`, vous n'avez pas besoin de faire référence au document qui contient le calque. Les scripts suivants créent un bloc de texte dans `myLayer`. Notez que, lorsque vous utilisez `myLayer`, vous n'avez pas besoin de fournir d'informations de hiérarchie de confinement relatives au calque.

REMARQUE : Le script suivant utilise la propriété `contents` pour ajouter du texte au bloc. Pour l'instant, ne vous souciez pas du mécanisme d'utilisation des propriétés.

Le script suivant utilise des objets et des propriétés définis dans le modèle objet Illustrator CS6 ; il ne fonctionne donc pas pour, par exemple, InDesign ou Photoshop.

AS

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc
    set myTextFrame to make text frame in myLayer
    set contents of myTextFrame to "Hello world!"
end tell
```

JS

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
var myTextFrame = myLayer.textFrames.add()
    myTextFrame.contents = "Hello world!"
```

VBS

```
Set appRef = CreateObject("Illustrator.Application")
Set docRef = appRef.Documents.Add
Set layerRef = docRef.Layers.Add
Set frameRef = layerRef.TextFrames.Add
    myTextFrame.Contents = "Hello world!"
```

Dénomination des variables

Vos scripts seront plus faciles à lire si vous créez des noms descriptifs pour vos variables. Des noms de variable comme `x` ou `c` n'aident pas beaucoup lorsque vous réétudiez un script. Les noms qui indiquent les données que la variable contient sont plus appropriés, comme `theDocument` ou `myLayer`.

Le fait de donner à vos noms de variable un préfixe standard permet de différencier les variables des objets, commandes et mots-clés de votre système d'écriture de scripts. Par exemple :

- Vous pouvez utiliser le préfixe « doc » au début de chaque variable qui contient des objets `Document`, par exemple `docRef` ou « layer » pour identifier les variables qui contiennent les objets `Art Layer`, par exemple `layerRef` et `layerRef2`.
- Vous pouvez utiliser le préfixe « my » pour ajouter un élément personnel qui distingue vos variables des objets de script. Par exemple, `myDoc` ou `myLayer` ou `myTextFrame`.

Tous les noms de variable doivent se conformer aux règles suivantes :

- Les noms de variable doivent être composés d'un seul mot (sans espace). De nombreuses personnes utilisent des majuscules (par exemple `myFirstPage`) ou des tirets longs (`my_first_page`) pour créer des noms plus lisibles. Le nom de la variable ne peut pas commencer par un tiret long.
- Les noms de variable peuvent contenir des chiffres mais ne peuvent pas commencer par un chiffre.
- Ils ne peuvent pas contenir de guillemets ou de marques de ponctuation autres que le tiret long.
- Les noms de variable en JavaScript et VBScript sont sensibles à la casse. `thisString` est différent de `thisstring` ou `ThisString`. Les noms de variable en AppleScript ne sont pas sensibles à la casse.
- Chaque variable de votre script doit avoir un nom unique.

Ensembles ou éléments d'objets comme références d'objet

Les langages de script placent chaque objet dans un *ensemble* (JS ou VBS) ou un *élément* (AS), puis affectent un numéro à l'objet appelé l'*indice* au sein de l'élément ou de l'ensemble. Les objets d'un élément ou d'un ensemble sont des types identiques d'objets. Par exemple, chaque objet `channel` de votre document appartient à un élément ou un ensemble `channels` ; chaque objet `art layer` appartient à un élément `art layers` ou à un ensemble `artLayers`.

En français, vous pouvez faire référence à un document en disant « Donnez-moi le premier document de l'ensemble ». Les langages de script vous permettent d'identifier un objet de la même manière, en utilisant son nom d'élément ou d'ensemble et son indice.

- En AS, vous faites référence au premier document de l'élément `documents` en tant que `document 1`.
- En JS, le premier document est `documents[0]`, (remarquez les crochets entourant l'indice) car (et c'est difficile de s'en souvenir au début) JavaScript commence à numéroter les objets d'un ensemble par 0.
- En VBS, le premier document est `Documents(0)`, (remarquez la parenthèse autour de l'indice). VBS commence la numérotation des objets d'un ensemble par 1.

Les scripts suivants référencent les objets `document` et `layer` par indice pour l'ajout de nouveaux objets.

REMARQUE : Comme le script suivant n'utilise pas de variables, la totalité de la hiérarchie de confinement est requise dans chaque référence d'objet. Par exemple, dans l'instruction qui ajoute un calque, le script doit identifier le document auquel le calque sera ajouté. Pour ajouter un bloc de texte au calque, le script doit fournir l'indice non seulement de la couche qui contiendra le bloc mais il doit également identifier le document qui contient le calque.

AS

```
tell application "Adobe InDesign CS6"
    make document
    make layer in document 1
    make text frame in layer 1 of document 1
end tell
```

REMARQUE : Les rédacteurs de script débutants utilisant AppleScript ne sont pas incités à utiliser les numéros d'élément comme références d'objet lorsque l'élément contient plus d'un objet. Pour plus d'explications, reportez-vous à « Comment les éléments et les ensembles numérotent les éléments suivants ? », page 14.

JS

En JavaScript, vous indiquez l'indice d'un élément en utilisant le nom de l'ensemble suivi par l'indice entre crochets (`[]`).

```
app.documents.add()
app.documents[0].layers.add()
app.documents[0].layers[0].textFrames.add()
```

REMARQUE : Souvenez-vous, en langage JS, les numéros d'indice d'un ensemble commencent par 0.

VBS

En VBScript, vous indiquez l'indice d'un élément en utilisant le nom de l'ensemble suivi de l'indice entre parenthèses.

```
appRef.Documents.Add
appRef.Documents(1).Layers.Add
appRef.Documents(1).Layers(1).TextFrames.Add
```

Comment les éléments et les ensembles numérotent les éléments suivants ?

Voici comment les langages de script gèrent la numérotation automatique si vous ajoutez un second objet à un ensemble ou un élément :

- AS affecte le numéro 1 au nouvel objet et renumérote l'objet déjà existant qui devient le numéro 2. Les numéros d'objet AppleScript parcourent les objets pour indiquer l'objet sur lequel vous avez travaillé le plus récemment. Ce processus peut rendre les longs scripts confus. En conséquence, nous encourageons les rédacteurs débutants à utiliser des variables comme références d'objet et à éviter l'utilisation d'indices.

- Les numéros d'ensemble JS sont statiques ; ils ne changent pas lorsque vous ajoutez un nouvel objet à l'ensemble. La numérotation des objets sous JS indique l'ordre dans lequel les objets ont été ajoutés à l'ensemble. Comme le premier objet que vous avez ajouté a reçu le numéro 0, l'objet suivant que vous ajoutez à l'ensemble porte le numéro 1 ; si vous ajoutez un troisième objet, il portera le numéro 2. Par exemple, lorsque vous ajoutez un document, le document contient automatiquement un calque. L'indice du calque est [0]. Si vous ajoutez un calque, l'indice du nouveau calque est [1] ; si vous ajoutez un second calque, son indice est [2]. Si vous faites glisser le calque [2] en haut de la palette Calques, il aura toujours l'indice [2].

- Les numéros d'ensemble VBS sont également statiques et la numérotation s'effectue exactement comme pour les ensembles JS, à la différence près que le premier objet de l'ensemble est toujours (1) en VBS.

CONSEIL : Dans les scripts JS et VBS, vous trouverez les numéros d'indice très utiles comme références d'objet. Par exemple, vous avez peut-être plusieurs fichiers pour lesquels vous souhaitez créer un calque d'arrière-plan blanc. Vous pouvez rédiger un script qui dit « Ouvrir tous les fichiers de ce dossier et changer la couleur du premier calque en blanc ». Si vous n'avez pas la possibilité de faire référence aux calques par indice, vous devez inclure dans votre script les noms de tous les calques d'arrière-plan dans tous les fichiers.

REMARQUE : Les scripts sont des organisateurs compulsifs. Ils placent les objets dans des éléments ou des ensembles même s'il n'y a qu'un seul objet de ce type dans la totalité de l'ensemble.

REMARQUE : Les objets peuvent appartenir à plus d'un ensemble ou d'un élément. Par exemple, sous Photoshop, les objets `art layer` appartiennent à l'élément ou à l'ensemble `art layers` et les objets `layer set` appartiennent à l'élément ou à l'ensemble `layer sets`, mais les objets `art layer` et `layer set` appartiennent tous à l'élément ou à l'ensemble `layers`. De même, sous InDesign, les objets `rectangle` appartiennent à l'élément ou à l'ensemble `rectangles` et les objets `text frame` appartiennent à l'élément ou à l'ensemble `text frames`. Néanmoins, à la fois les objets `rectangle` et les objets `text frame` appartiennent également à l'élément ou à l'ensemble `page items` qui contient toutes sortes d'éléments sur une page, comme des ellipses, des lignes de graphique, des polygones, des boutons et d'autres éléments.

Référence à l'objet actuel ou actif

Lorsque vous avez exécuté votre premier script et créé un nouveau document, l'application s'est ouverte, puis a créé un document. Si vous vouliez modifier ce document dans l'interface utilisateur de l'application, vous auriez pu simplement travailler avec la souris, les menus, la boîte à outils et les palettes car le document a été automatiquement sélectionné.

Cette assertion est vraie pour tous les objets que vous créez dans un script. Jusqu'à ce que le script fasse quelque chose d'autre, le nouvel objet est l'objet actif, prêt à être modifié.

De façon pratique, de nombreux objets parents contiennent des propriétés qui vous permettent de vous référer facilement à l'objet actif. (Vous en apprendrez plus sur les propriétés un peu plus loin dans ce guide. Pour l'instant, vous pouvez simplement copier les instructions de script de cette section et observer comment elles fonctionnent sans comprendre complètement pourquoi elles apparaissent de la sorte.)

- En AS, la propriété qui fait référence à un objet actif est composée du mot `current` et du nom de l'objet. Quelques exemples :

```
current document
current layer
current channel
current view
```

- En JS, le nom de la propriété est un mot composé de `active` et du nom de l'objet, selon l'utilisation de la casse standard de JS :
  - Le premier mot est en minuscules.
  - Le second mot (et tous ceux qui suivent) utilisent des majuscules pour la première lettre.

  Quelques exemples :

```
activeDocument
activeLayer
activeChannel
activeView
```

- VBS est identique à JS sauf que tous les mots du terme composé utilisent une majuscule pour la première lettre. Quelques exemples :

```
ActiveDocument
ActiveLayer
ActiveChannel
ActiveView
```

Les scripts suivants créent un document, puis utilisent ce principe pour créer un calque dans le nouveau document.

AS

```
tell application "Adobe Illustrator CS6"
    make document
    make layer in current document
end tell
```

JS

```
app.documents.add()
app.activeDocument.layers.add()
```

REMARQUE : Vérifiez que vous avez saisi `activeDocument` sans `s` à la fin.

VBS

```
Set appRef = CreateObject("Illustrator.Application")
docRef.Documents.Add
appRef.ActiveDocument.Layers.Add
```

REMARQUE : Vérifiez que vous avez saisi `ActiveDocument` sans `s` à la fin.

Utilisation des propriétés

Pour définir ou modifier une propriété d'un objet, vous devez faire trois choses :

1. Nommer l'objet.
2. Nommer la propriété.
3. Indiquer la valeur de la propriété.

La valeur peut être un des types de données suivants (n'importe lequel) :

- Une chaîne, qui est du texte alphanumérique, qui est interprétée comme du texte. Les chaînes doivent être entourées de guillemets (" "). Les chaînes incluent des valeurs comme le nom d'un objet.
- Un nombre qui est une valeur numérique pouvant être utilisée dans des opérations mathématiques comme l'addition ou la division. Les nombres mathématiques incluent la longueur d'un côté d'un bloc ou l'espace entre des paragraphes, le pourcentage d'opacité, la taille de police, l'épaisseur du contour, etc.

  Notez que certaines valeurs qui ressemblent à des nombres sont en fait des chaînes. Par exemple, un numéro de téléphone ou de sécurité sociale sont des nombres mais vous les formateriez comme chaînes (entre guillemets) car les données ne seraient pas considérées comme des nombres mathématiques.

Au sein de la catégorie numérique, il existe différents types de nombres :

- ▷ Nombre entier qui est un nombre sans décimale.
- ▷ Nombres réel, fixe, court, long ou double qui sont des numéros qui peuvent inclure des décimales, comme 5,9 ou 1,0.

Remarque :Ces différences peuvent sembler futiles mais gardez-les à l'esprit pour plus tard.

- ▶ Une variable. Lorsque vous utilisez une variable comme valeur de la propriété, vous ne placez pas la variable entre des guillemets comme vous le feriez pour une chaîne.
- ▶ Une valeur booléenne, qui est soit `true` ou `false`.

REMARQUE : Souvent, les valeurs booléennes agissent comme des interrupteurs.

- ▶ Une valeur constante (également appelée *enumeration*) qui est un ensemble prédéfini de valeurs dans lequel vous pouvez effectuer une sélection. L'utilisation de valeurs constantes pour une propriété est similaire en termes de concept à l'utilisation d'un menu déroulant dans une application Adobe. Les constantes ainsi que leur mode d'utilisation sont présentés dans « Valeurs constantes et énumérations », page 21.
- ▶ Une liste (AS) ou un tableau (JS et VBS).

  Certaines propriétés requièrent plusieurs valeurs comme les coordonnées d'emplacement d'un point (coordonnées x et y) ou les limites d'un bloc de texte ou d'un objet géométrique. Des valeurs multiples pour une seule propriété sont appelées liste en AS et tableau en JS ou VBS. Chaque langage spécifie des règles de formatage.

  - ▷ La liste ou le tableau doit être présenté(e) comme suit :

    En AS, la liste est entourée d'accolades : `{}`

    En JS, le tableau est entouré de crochets : `[]`

    En VBS, le tableau est entouré de parenthèses et suit le mot-clé `Array`: `Array()`

  - ▷ Les valeurs sont séparées par une virgule (`,`). Vous pouvez inclure ou omettre des espaces après les virgules ; cela n'a pas d'importance.

    AS `{3,4,5}` ou `{"string1", "string2", "string3"}`

    JS `[3,4,5]` ou `["string1", "string2", "string3"]`

    VBS `Array(3,4,5)` ou `Array("string1", "string2", "string3")`

  - ▷ Une liste ou un tableau peut inclure des listes ou des tableaux imbriqués, comme une liste de coordonnées. Dans les exemples suivants, remarquez que chaque tableau imbriqué est entouré individuellement d'accolades, de parenthèses ou de crochets et que les tableaux imbriqués sont séparés par des virgules.

    AS `{{x1, y1}, {x2, y2}, {x3, y3}}`

    JS `[[x1, y1], [x2, y2], [x3, y3]]`

    VBS `Array(Array(x1, y1), Array(x2, y2), Array(x3, y3))`

AS

Pour utiliser les propriétés en langage AS, utilisez la commande `set` suivie du nom de la propriété, puis saisissez `of` suivi de la référence d'objet. Le script suivant définit la propriété `name` de l'objet `layer`.

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc
    set name of myLayer to "My New Layer"
end tell
```

Vous pouvez définir plusieurs propriétés dans une seule instruction en utilisant la propriété `properties`. Vous formatez les propriétés multiples dans un tableau, entouré d'accolades. Dans le tableau, séparez chaque paire de nom/valeur de propriété par deux points (`:`). Le script suivant utilise `properties` pour définir le nom du calque et l'état de visibilité.

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc
    set properties of myLayer to {name:"My New Layer", visible:false}
end tell
```

REMARQUE : Notez que, dans le script ci-dessus, seule la valeur de la chaîne `"Mon nouveau calque"` est entre guillemets. La valeur de la propriété `visible`, `false` peut ressembler à une chaîne mais il s'agit d'une valeur booléenne. Pour consulter les types de valeurs, reportez-vous à « Utilisation des propriétés », page 16.

Vous pouvez définir les propriétés d'un objet dans l'instruction qui crée l'objet, comme dans les scripts ci-dessous.

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc with properties {name:"My New Layer"}
end tell
```

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc with properties {name:"My New Layer",
       visible:false}
end tell
```

JS

Pour utiliser une propriété en langage JS, nommez l'objet que vous souhaitez que la propriété définisse ou modifie, insérez un point (`.`), puis nommez la propriété. Pour indiquer la valeur, placez un symbole égal (`=`) après le nom de la propriété, puis saisissez la valeur.

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
    myLayer.name = "My New Layer"
```

Pour définir des propriétés multiples, vous pouvez rédiger plusieurs instructions :

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
    myLayer.name = "My New Layer"
    myLayer.visible = false
```

REMARQUE : Notez que, dans le script ci-dessus, seule la valeur de la chaîne `"Mon nouveau calque"` est entre guillemets. La valeur de la propriété `visible`, `false` peut ressembler à une chaîne mais il s'agit d'une valeur booléenne. Pour consulter les types de valeurs, reportez-vous à « Utilisation des propriétés », page 16.

JS fournit un abrégé pour définir des propriétés multiples appelé instruction `with`. Pour utiliser une instruction `with`, utilisez le mot `with` suivi de l'objet dont vous souhaitez définir les propriétés, en entourant la référence d'objet de parenthèses (`()`). Ne saisissez pas d'espace entre `with` et la première parenthèse. Ensuite, saisissez une accolade ouverte (`{`), puis appuyez sur **Entrée** et saisissez un nom et une valeur de propriété sur la ligne suivante. Pour fermer l'instruction `with`, saisissez une accolade fermée (`}`).

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
    with(myLayer){
    name = "My New Layer"
    visible = false
    }
```

L'utilisation d'une instruction `with` vous épargne d'avoir à saisir la référence d'objet suivie d'un point (dans ce cas, `myLayer.`) pour chaque propriété. Lorsque vous utilisez une instruction `with`, n'oubliez jamais l'accolade fermée.

JS fournit également une propriété `properties` qui vous permet de définir plusieurs valeurs dans une seule instruction. Incluez l'ensemble du groupe de valeurs dans des accolades (`{}`). A l'intérieur des accolades, utilisez deux points (`:`) pour séparer un nom de propriété de sa valeur et séparez les paires nom/valeur de propriété à l'aide d'une virgule (`,`).

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
    myLayer.properties = {name:"My New Layer", visible:false}
```

VBS

Pour utiliser des propriétés en langage VBS, nommez l'objet, insérez un point (`.`), puis nommez la propriété. Pour indiquer la valeur, placez un symbole égal (=) après le nom de la propriété, puis saisissez la valeur.

```
Set appRef = CreateObject("Illustrator.Application")
Set myDoc = appRef.Documents.Add
Set myLayer = myDoc.Layers.Add
    myLayer.Name = "My First Layer"
```

Vous ne pouvez définir qu'une seule propriété par instruction. Pour définir des propriétés multiples, vous devez rédiger plusieurs instructions :

```
Set appRef = CreateObject("Illustrator.Application")
Set myDoc = appRef.Documents.Add
Set myLayer = myDoc.Layers.Add
    myLayer.Name = "My First Layer"
    myLayer.Opacity = 65
    myLayer.Visible = false
```

REMARQUE : Notez que, dans le script ci-dessus, seule la valeur de la chaîne `"Mon nouveau calque"` est entre guillemets. La valeur de la propriété `Visible`, `false` peut ressembler à une chaîne mais il s'agit d'une valeur booléenne. Pour consulter les types de valeurs, reportez-vous à « Utilisation des propriétés », page 16.

Utilisation des propriétés lecture seule et lecture-écriture

Lors de la définition des valeurs de propriété, vous pouvez rédiger une instruction de script avec une syntaxe parfaite mais l'instruction ne produit aucun résultat. Cela peut se produire lorsque vous tentez de définir une propriété qui n'est pas « en écriture » ; la propriété est *en lecture seule*.

Par exemple, la propriété `name` de l'objet document de la plupart des applications Adobe est en lecture seule ; en conséquence, vous ne pouvez pas utiliser un script pour définir ou modifier le nom d'un document existant (bien que vous puissiez utiliser une commande ou méthode `save as` ; reportez-vous à « Utilisation de méthodes ou commandes », page 24 pour plus d'informations). Alors pourquoi se préoccuper d'avoir une propriété que vous ne pouvez pas définir, pourriez-vous demander. La réponse est que les propriétés en lecture seule sont des sources non négligeables d'informations. Par exemple, vous pouvez vouloir connaître quel est le nom d'un document ou combien de documents figurent dans l'ensemble `Documents`.

Utilisation des avertissements de sécurité pour afficher la valeur d'une propriété

Une manière conviviale d'afficher les informations d'une propriété en lecture seule est d'utiliser l'avertissement de sécurité. Il s'agit d'une petite boîte de dialogue qui affiche simplement des informations. Vous pouvez utiliser les avertissements de sécurité pour afficher la valeur de n'importe quelle propriété : lecture-écriture ou lecture seule.

AS

Pour afficher un avertissement de sécurité en AS, saisissez `display dialog`, puis entrez le contenu de la boîte de dialogue entre parenthèses (`()`). Pour savoir combien d'objets figurent dans un élément, utilisez la commande `count` avec n'importe quel nom d'élément.

REMARQUE : Le nom d'élément est la forme plurielle de l'objet. Par exemple, l'élément de l'objet `document` est l'objet `documents`.

Le script suivant affiche un avertissement de sécurité qui vous indique combien de documents figurent dans l'élément `documents`, puis ajoute un document et affiche une nouvelle alerte avec le nombre mis à jour.

```
tell application "Adobe Photoshop CS6"
    display dialog (count documents)
    set myDoc to make document
    display dialog (count documents)
end tell
```

Pour qu'une valeur de chaîne s'affiche dans un avertissement de sécurité, vous devez stocker la valeur de chaîne dans une variable. Le script suivant convertit le nom du document en une variable nommée `myName`, puis affiche la valeur de `myName`.

```
tell application "Adobe Photoshop CS6"
    set myDoc to make document
    set myName to name of myDoc
    display dialog myName
end tell
```

JS

Pour afficher un avertissement de sécurité en JS, utilisez la méthode `alert()` en saisissant `alert`, puis en entrant le contenu de la boîte de dialogue entre parenthèses (`()`). Ne saisissez pas d'espace entre `alert` et la première parenthèse. Pour savoir combien d'objets figurent dans un ensemble, utilisez la propriété `length` (en lecture seule) de n'importe quel objet de l'ensemble. Le script suivant affiche un avertissement de sécurité qui vous indique combien de documents figurent dans l'ensemble `documents`, puis ajoute un document et affiche une nouvelle alerte avec le nombre mis à jour.

REMARQUE : Le nom de l'ensemble est la forme plurielle de l'objet. Par exemple, l'objet d'ensemble de l'objet `document` est l'objet `documents`.

```
alert(app.documents.length)
var myDoc = app.documents.add()
alert(app.documents.length)
```

Le script suivant affiche le nom du document dans un avertissement de sécurité.

```
var myDoc = app.documents.add()
alert(myDoc.name)
```

VBS

Pour afficher un avertissement de sécurité en VBS, utilisez la méthode `MsgBox` en saisissant `MsgBox`, puis en entrant le contenu de la boîte de dialogue entre parenthèses (`()`). Ne saisissez pas d'espace entre `MsgBox` et la première parenthèse. Pour savoir combien d'objets figurent dans un ensemble, utilisez la propriété `Count` (en lecture seule) de n'importe quel objet de l'ensemble. Le script suivant affiche un avertissement de sécurité qui vous indique combien de documents figurent dans l'ensemble `Documents`, puis ajoute un document et affiche une nouvelle alerte avec le nombre mis à jour.

REMARQUE : L'ensemble de l'objet est la forme plurielle de l'objet. Par exemple, l'objet d'ensemble de l'objet `Document` est l'objet `Documents`.

```
Set appRef = CreateObject("Photoshop.Application")
MsgBox(appRef.Documents.Count)
Set myDoc = appRef.Documents.Add
    MsgBox(appRef.Documents.Count)
```

Le script suivant affiche le nom du document dans un avertissement de sécurité.

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Documents.Add
    MsgBox(myDoc.Name))
```

Valeurs constantes et énumérations

Certaines valeurs de propriétés sont prédéfinies par l'application. Par exemple, dans la plupart des applications, l'orientation de la page peut être paysage ou portrait. L'application n'accepte que l'une de ces deux valeurs. Elle n'accepte pas « vertical », « droit », « horizontal » ou « sur le côté ». Pour garantir que votre script fournit une valeur acceptable pour la propriété d'orientation de page d'un document, la propriété a été écrite pour n'accepter qu'une valeur prédéfinie.

Dans l'écriture de scripts, ces valeurs prédéfinies sont appelées *constantes* ou *énumérations*.

L'utilisation d'une constante ou d'une énumération est similaire à l'utilisation d'une liste déroulante dans l'interface utilisateur d'une application.

REMARQUE : Pour savoir si vous devez utiliser une énumération pour une valeur de propriété, recherchez la propriété dans l'une des références de l'écriture de scripts fournies par Adobe. Pour plus d'informations, reportez-vous au Chapitre 3, « Recherche de propriétés et de méthodes d'objets ».

AS

En langage AS, vous utilisez des constantes comme vous le feriez avec n'importe quelle autre définition de propriété. N'entourez pas la constante de guillemets. Le script suivant utilise la valeur de constante `dark green` pour définir la couleur d'un nouveau calque.

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc
    set layer color of myLayer to dark green
end tell
```

REMARQUE : Si `dark green` était une valeur de chaîne et non une constante, elle serait entourée de guillemets :

```
    set layer color of myLayer to "dark green"
```

JS

En langage JS, saisissez le nom de l'énumération, un point (`.`), puis la valeur de l'énumération. Vous devez utiliser l'orthographe et l'emploi de majuscules exacts définis dans les références de l'écriture de scripts fournies par Adobe. Le formatage diffère selon les applications Adobe. Par exemple :

- Sous InDesign :
  - Chaque énumération commence par une lettre majuscule et tous les mots du terme composé commencent également par une lettre majuscule.
  - La valeur d'énumération commence par une minuscule.

    L'exemple suivant utilise l'énumération `UIColor` pour définir la couleur du calque sur vert foncé.

    ```
    var myDoc = app.documents.add()

    var myLayer = mydoc.layers.add()

        myLayer.layerColor = UIColor.darkGreen
    ```

- Sous Illustrator :
  - Chaque énumération commence par une lettre majuscule et tous les mots du terme composé commencent également par une lettre majuscule.
  - Certaines valeurs d'énumération commencent par une lettre majuscule, puis utilisent des lettres minuscules. Les autres utilisent uniquement des majuscules. Vous devez vous assurer d'utiliser la valeur exactement comme elle apparaît dans la référence de l'écriture de scripts.

    L'exemple suivant utilise l'énumération `RulerUnits` pour définir l'unité par défaut sur centimètres.

    ```
    var myDoc = app.documents.add()

        myDoc.rulerUnits = RulerUnits.Centimeters
    ```

Le script suivant utilise l'énumération `BlendModes` dont les valeurs sont exprimées en lettres majuscules uniquement.

```
var myDoc = app.documents.add()

var myLayer = myDoc.layers.add()

    myLayer.blendingMode = BlendModes.COLORBURN
```

- Sous Photoshop :
  - Chaque énumération commence par une lettre majuscule et tous les mots du terme composé commencent également par une lettre majuscule.
  - Les valeurs d'énumération sont uniquement constituées de majuscules.

    L'exemple suivant utilise l'énumération `LayerKind` pour faire du calque un calque de texte.

```
var myDoc = app.documents.add()
var myLayer = mydoc.artLayers.add()

    myLayer.kind = LayerKind.TEXT
```

VBS

En langage VBS, vous utilisez des valeurs numériques pour les constantes.

```
Set appRef = CreateObject("Photoshop.Application")
Set docRef = appRef.Documents.Add
Set layerRef = docRef.ArtLayers.Add
    layerRef.Kind = 2
```

Utilisation de variables pour les valeurs de propriété

Vous pouvez utiliser des variables pour contenir des valeurs de propriété. Cela peut vous aider à mettre à jour un script rapidement et avec précision. Par exemple, vous avez une publication dont toutes les photos sont au format 10 x 15 cm. Si vous utilisez une variable pour définir la hauteur et la largeur et que les mesures changent, vous n'avez qu'à modifier les valeurs d'une variable au lieu de modifier les mesures de chaque photo du document.

Le script suivant crée des variables pour contenir les valeurs de largeur et de hauteur du document, puis utilise les variables comme valeurs dans l'instruction qui modifie la largeur et la hauteur.

AS

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document with properties {height:10, width:7}
    set docHeight to height of myDoc
    set docWidth to width of myDoc
    set myDoc with properties {height:docHeight - 2, width:docWidth - 2}
end tell
```

JS

```
var myDoc = app.documents.add(7, 10)
var docHeight = myDoc.height
var docWidth = myDoc.width
       myDoc.resizeCanvas((docHeight - 2), (docWidth - 2))
```

VBS

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Documents.Add(7, 10)
docHeight = myDoc.Height
docWidth = myDoc.Width
myDoc.ResizeCanvas docWidth - 2, docHeight - 2
```

REMARQUE : La méthode `MsgBox` ne fonctionne pas lorsque vous ouvrez un script dans le menu Scripts de certaines applications Adobe. Pour afficher correctement la boîte de dialogue de message, double-cliquez sur le fichier de script dans Windows Explorer®.

Utilisation de méthodes ou commandes

Les commandes (en AS) et les méthodes (en VBS et JS) sont des instructions que vous ajoutez à un script pour effectuer des tâches ou obtenir des résultats. Par exemple, vous pouvez utiliser la commande/méthode `print/print()/PrintOut` pour imprimer un document.

AS Les commandes AS apparaissent au début d'une instruction de script comme un verbe à l'impératif. La commande est suivie par une référence à l'objet sur lequel vous souhaitez que la commande agisse.

Le script suivant imprime le document actif :

```
tell application "Adobe InDesign CS6"
    print current document
end tell
```

JS Vous insérez des méthodes à la fin des instructions JS. Vous devez placer un point avant le nom de la méthode, puis vous indiquez le nom de la méthode entre parenthèses (`()`).

```
app.activeDocument.print()
```

VBS Vous insérez des méthodes à la fin des instructions VBS. Vous devez placer un point avant le nom de la méthode.

```
Set appRef = CreateObject("Photoshop.Application")
appRef.ActiveDocument.PrintOut
```

Paramètres des commandes ou méthodes

Certaines commandes ou méthodes requièrent des données supplémentaires, appelées *arguments ou paramètres*. Les commandes ou les méthodes peuvent également inclure des paramètres supplémentaires.

Paramètres requis

Les scripts suivants utilisent la commande `merge` qui requiert des indications relatives aux calques que vous souhaitez fusionner dans le calque sélectionné. Comme les propriétés, les paramètres des commandes sont entourés d'accolades (`{}`). Néanmoins, n'incluez que la valeur du paramètre, et non le nom du paramètre, entre les accolades.

REMARQUE : Ce script est conçu pour InDesign. Il n'existe pas d'opération de fusion dans Illustrator. Pour modifier ce script pour Photoshop, notez qu'un calque est appelé `art layer` en AS et les calques sont appelés `artLayers` en JS ou `ArtLayers` en VBS).

AS

```
tell application "Adobe InDesign CS6"
    set myDoc to make document

    set myLayer to make layer in myDoc
    set myLayer2 to make layer in myDoc

    merge myLayer2 with {myLayer}
end tell
```

JS

Le paramètre de la méthode est entouré de parenthèses qui suivent le nom de la méthode.

```
var myDoc = app.documents.add()

var myLayer = myDoc.layers.add()
var myLayer2 = myDoc.layers.add()

myLayer2.merge(myLayer)
```

VBS

Notez que le paramètre de méthode est entouré de parenthèses après le nom de la méthode. Ne saisissez pas d'espace avant la première parenthèse.

```
Set appRef = CreateObject("InDesign.Application")
Set myDoc = appRef.Documents.Add

Set myLayer = myDoc.Layers.Add
Set myLayer2 = myDoc.Layers.Add

myLayer2.Merge(myLayer)
```

Paramètres multiples

Lorsque vous définissez plus d'un paramètre pour une commande ou une méthode, vous devez suivre des règles spécifiques.

AS

Il existe deux types de paramètres pour les commandes AS :

- Un paramètre *direct* qui définit l'objet direct de l'action exécutée par la commande
- Des paramètres *libellés* qui sont tous les paramètres autres que les paramètres directs

Le paramètre direct doit suivre directement la commande. Dans l'instruction suivante, la commande est `make` et le paramètre direct est `document`.

```
make document
```

Vous pouvez insérer des paramètres libellés dans n'importe quel ordre. Le script suivant crée deux calques et définit l'emplacement et le nom de chaque couche. Notez que, dans les instructions qui créent les calques, les paramètres `location` et `name` apparaissent dans des ordres différents.

```
tell application "Adobe InDesign CS6"
    set myDoc to make document
    tell myDoc
        set myLayer to make layer at beginning of myDoc with properties {name:"Lay1"}
        set myLayer2 to make layer with properties {name:"Lay2"} at end of myDoc
    end tell
end tell
```

JS

En JS, vous devez saisir des valeurs de paramètres dans l'ordre dans lequel ils sont répertoriés dans les ressources de référence de l'écriture de scripts afin que le compilateur de scripts connaisse quelle valeur définit quel paramètre.

REMARQUE : Pour plus d'informations sur les ressources de référence de l'écriture de scripts, reportez-vous au Chapitre 3, « Recherche de propriétés et de méthodes d'objets ».

Pour ignorer un paramètre en option, saisissez la balise d'emplacement `undefined`. L'instruction suivante crée un document Photoshop CS6 dont la largeur est 4 000 pixels, la hauteur 5 000 pixels, la résolution 72, le nom « Mon document ». Le mode du document est bitmap.

```
app.documents.add(4000, 5000, 72, "My Document", NewDocumentMode.BITMAP)
```

L'instruction suivante crée un document identique sauf que la résolution n'est pas définie.

```
app.documents.add(4000, 5000, undefined, "My Document", NewDocumentMode.BITMAP)
```

REMARQUE : Utilisez la balise d'emplacement `undefined` uniquement pour « atteindre » les paramètres à définir. L'instruction suivante définit uniquement la hauteur et la largeur du document ; les balises d'emplacement ne sont pas nécessaires pour les paramètres en option suivants.

```
    app.documents.add(4000, 5000)
```

VBS

En VBS, vous devez saisir des valeurs de paramètres dans l'ordre dans lequel ils sont répertoriés afin que le compilateur de scripts connaisse quelle valeur définit quel paramètre.

Pour ignorer un paramètre en option, saisissez la balise d'emplacement `undefined`. L'instruction suivante crée un document Photoshop CS6 dont la largeur est 4 000 pixels, la hauteur 5 000 pixels, la résolution 72, le nom « Mon document ». Le mode du document est bitmap.

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Documents.Add(4000, 5000, 72, "My Document", 5)
```

L'instruction suivante crée un document identique sauf que la résolution n'est pas définie.

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Documents.Add(400, 500, undefined, "My Document", 5)
```

REMARQUE : Utilisez la balise d'emplacement `undefined` uniquement pour « atteindre » les paramètres à définir. L'instruction suivante définit uniquement la hauteur et la largeur du document ; les balises d'emplacement ne sont pas nécessaires pour les paramètres en option suivants.

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Documents.Add(4000, 5000)
```

La balise d'emplacement `undefined` n'est pas sensible à la casse.

Instructions Tell (AS uniquement)

Vous avez peut-être remarqué que les exemples AppleScript commencent et se terminent par les instructions :

```
tell application "Nom de l'application"
end tell
```

L'instruction `tell` nomme l'objet par défaut qui exécute toutes les commandes contenues dans l'instruction. Dans l'exemple précédent, l'instruction `tell` cible l'objet application. En conséquence, toute commande contenue dans l'instruction doit être exécutée par l'objet application sauf si un autre objet est explicitement nommé dans une instruction de script dans l'instruction `tell`.

Le script suivant présente clairement la totalité de la hiérarchie de confinement de chaque objet afin d'indiquer l'objet ciblé par la commande :

```
tell application "Adobe InDesign CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc
    set myLayer2 to make layer in myDoc
end tell
```

Vous pouvez créer un raccourci en changeant la cible de la commande. Pour ce faire, ajoutez une instruction `tell` imbriquée. Le script suivant effectue exactement la même opération que le script précédent. Puisque l'instruction `tell` imbriquée cible l'objet document, il n'est pas nécessaire de faire référence à l'objet document dans les instructions qui créent les calques.

```
tell application "Adobe InDesign CS6"
    set myDoc to make document
    tell myDoc
        set myLayer to make layer
        set myLayer2 to make layer
    end tell
end tell
```

Notez que chaque instruction `tell` doit être incluse dans sa propre instruction `end tell`.

Vous pouvez imbriquer autant d'instructions `tell` que vous le souhaitez.

Remarques sur les variables

Cette section fournit des informations supplémentaires sur l'utilisation des variables.

Modification de la valeur d'une variable

Vous pouvez modifier la valeur d'une variable à n'importe quel moment. Pour ce faire, utilisez simplement le nom de la variable suivi de l'opérateur d'affectation (`to` en AS ; `=` en JS ou VBS) et de la nouvelle valeur. Les scripts suivants créent la variable `layerRef` pour contenir un nouveau calque, puis créent immédiatement un second calque et l'affectent comme nouvelle valeur de `layerRef`.

AS Pour modifier la valeur d'une variable en AS, utilisez la commande `set`.

```
tell application "Adobe Illustrator CS6"
    set docRef to make document
    set layerRef to make layer in myDoc with properties {name:"First Layer"}
    set layerRef to make layer in myDoc with properties {name:"Second Layer"}
end tell
```

JS Pour modifier la valeur d'une variable en JS, utilisez le nom de la variable suivi du symbole égal (=) et de la nouvelle valeur. Ne commencez pas l'instruction de réaffectation par `var` ; utilisez `var` uniquement lorsque vous créez une nouvelle variable.

```
var docRef = app.documents.add()
var layerRef = myDoc.layers.add()
    layerRef.name = "First Layer"
    layerRef = myDoc.layers.add()
    layerRef.name = "Second Layer"
```

VBS Pour modifier la valeur d'une variable en VBS, utilisez la commande `Set`.

```
Set appRef = CreateObject("Illustrator.Application")
Set docRef = appRef.Documents.Add
Set layerRef = docRef.Layers.Add
    layerRef.Name = "First Layer"
    layerRef = docRef.Layers.Add
    layerRef.Name = "Second Layer"
```

Utilisation de variables pour faire référence à des objets existants

Vous pouvez également créer des variables pour contenir des objets existants.

AS

```
tell application "Adobe Photoshop CS6"
    set myDoc to active document
end tell
```

JS

```
var myDoc = app.activeDocument
```

VBS

```
Set appRef = CreateObject("Illustrator.Application")
Set docRef = appRef.ActiveDocument
```

Rendre les fichiers de script lisibles

Cette section présente deux options qui vous aident à rendre les fichiers de script plus lisibles :

- Commentaires
- Sauts de ligne

Commenter le script

Un commentaire de script est un texte que le moteur de script ignore lorsqu'il exécute le script.

Les commentaires sont très utiles lorsque vous voulez documenter le fonctionnement ou l'objet d'un script (pour vous-même ou pour quelqu'un d'autre). La plupart des programmeurs, même les plus qualifiés, prennent du temps pour insérer des commentaires pour presque chaque élément d'un script. Les commentaires peuvent ne pas vous sembler importants lorsque vous rédigez vos scripts mais vous serez content d'avoir inclus des commentaires lorsque, un mois ou une année plus tard, vous ouvrirez un script et vous vous demanderez ce que vous aviez essayé de faire et pourquoi.

AS Pour commenter toute ou partie d'une ligne en AS, saisissez deux traits d'union (`--`) au début du commentaire. Pour commenter plusieurs lignes, entourez le commentaire par `(*` et `*)`.

```
tell application "Adobe InDesign CS6"
--This is a single-line comment
print current document --this is a partial-line comment
--the hyphens hide everything to their right from the scripting engine
(* This is a multi-line
   comment, which is completely
   ignored by the scripting engine, no matter how
   many lines it contains.
   The trick is to remember to close the comment.
   If you donít the rest of your script is
   hidden from the scripting engine!*)
end tell
```

REMARQUE : La seule chose que fait ce script est d'imprimer le document ouvert.

JS Pour commenter toute ou partie d'une ligne en JS, saisissez deux barres obliques (`//`) au début du commentaire. Pour commenter plusieurs lignes, entourez le commentaire de `/*` et `*/`.

```
//This is a single-line comment
app.activeDocument.print() //this part of the line is also a comment

/* This is a multi-line
   comment, which is completely
   ignored by the scripting engine, no matter how
   many lines it contains.
   Don?t forget the closing asterisk and slash
   or the rest of your script will be commented out...*/
```

REMARQUE : La seule chose que fait ce script est d'imprimer le document actif.

VBS En VBS, saisissez `Rem` (pour « remarque ») ou `'` (une apostrophe simple droite) au début du commentaire. VBS ne prend pas en charge les commentaires sur plusieurs lignes. Pour commenter plusieurs lignes en une seule fois, commencez chaque ligne avec l'un des deux formats de commentaire.

```
'This is a comment.
Set appRef = CreateObject("Photoshop.Application")
Rem This is also a comment.
appRef.ActiveDocument.PrintOut 'This part of the line is a comment.
' This is a multi-line
' comment that requires
' a comment marker at the beginning
' of each line.
Rem This is also a multi-line comment. Generally, multi-line
Rem comments in VBS are easier for you to identify (and read) in your scripts
Rem if they begin with a single straight quote (') rather than if they begin
Rem with Rem, because Rem can look like any other text in the script
' The choice is yours but isn?t this more easily
' identifiable as a comment than the preceding
' four lines were?
```

REMARQUE : La seule chose que fait ce script est d'imprimer le document actif.

Continuer des lignes longues en AppleScript et VBScript

En AppleScript et VBScript, un retour chariot à la fin d'une ligne signale la fin d'une instruction. Lorsque votre script est trop long pour tenir sur une ligne, vous pouvez utiliser des caractères spéciaux de *continuation* - caractères qui insèrent un saut de ligne, mais indiquent au script de lire la ligne coupée comme une seule instruction.

REMARQUE : Vous pouvez également étendre la fenêtre d'éditeur de scripts pour poursuivre l'instruction sur une seule ligne.

AS Tapez le caractère ¬ (**Option+Retour**) pour couper une longue ligne tout en continuant l'instruction.

```
tell application "Adobe InDesign CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc with properties {name:"Mon premier calque"} at
    the¨¬
    beginning of myDoc (* sans le caractère de saut de ligne, AS considérerait que cette
        ligne est une instruction incomplète*)
    (* notez que les caractères de continuation ne sont pas requis dans un commentaire sur
    plusieurs lignes
    tel que celui-ci*)
    set myLayer2 to make layer in myDoc with properties {name:"Mon autre calque"} ¨ ¬
    before myLayer
end tell
```

VBS Saisissez un tiret long (_) suivi d'un retour chariot pour couper une longue ligne tout en continuant l'instruction.

REMARQUE : Dans les deux langages, le caractère de continuation perd ses fonctionnalités s'il est placé à l'intérieur d'une chaîne (c'est-à-dire à l'intérieur de guillemets). Si le saut de ligne se produit au sein d'une valeur de chaîne, placez le caractère de saut avant la chaîne et insérez le saut de ligne avant.

REMARQUE : En JavaScript, les instructions peuvent contenir des retours chariot. Il n'est donc pas nécessaire d'utiliser un caractère de continuation. Néanmoins, l'interpréteur ExtendScript interprète chaque ligne comme une instruction complète. Ainsi, en général, il est préférable d'insérer des retours uniquement à la fin des instructions.

Utilisation de tableaux

En VBScript et JavaScript, les tableaux sont similaires à des ensembles. Néanmoins, à la différence des ensembles, les tableaux ne sont pas créés automatiquement.

Vous pouvez vous représenter un tableau comme une liste de valeurs pour une seule variable. Par exemple, le tableau JavaScript suivant répertorie quatre valeurs pour la variable `myFiles` :

```
var myFiles = new Array ()
    myFiles[0] = "clouds.bmp"
    myFiles[1] = "clouds.gif"
    myFiles[2] = "clouds.jpg"
    myFiles[3] = "clouds.pdf"
```

Remarquez que chaque valeur est numérotée. Pour utiliser une valeur dans une instruction, vous devez inclure le numéro. L'instruction suivante ouvre le fichier `clouds.gif`:

```
open(myFiles[1])
```

L'exemple suivant inclut les mêmes instructions en VBScript :

```
Dim myFiles (4)
    myFiles(0) = "clouds.bmp"
    myFiles(1) = "clouds.gif"
    myFiles(2) = "clouds.jpg"
    myFiles(3) = "clouds.pdf"
appRef.Open myFiles(1)
```

REMARQUE : Alors que les indices des ensembles VBS commencent toujours la numérotation par (1), vous pouvez indiquer dans vos scripts VBS si les tableaux que vous créez commencent la numérotation par (1) ou (0). Pour savoir comment définir le numéro de début d'indice d'un tableau, reportez-vous à la documentation VBScript. Pour plus d'informations sur les ensembles et les numéros d'indice, reportez-vous à « Ensembles ou éléments d'objets comme références d'objet », page 13.

Création d'objets

Votre premier script démontrait comment créer un objet en utilisant la commande `make` (AS), la méthode `add()` (JS) ou la méthode `Add` (VBS) de l'objet d'ensemble de l'objet. Par exemple :

AS

```
tell application "Adobe Photoshop CS6"
    make document
end tell
```

JS

```
app.documents.add()
```

VBS

```
Set appRef = CreateObject("Photoshop.Application")
appRef.Documents.Add()
```

Néanmoins, certains objets n'ont pas de commande `make` (AS), de méthode `add()` (JS) ou de méthode `Add` (VBS). Pour créer des objets de ce type, reportez-vous à la section Création d'objets du chapitre correspondant à votre langage de script dans le guide d'écriture de scripts Adobe de votre application.

Informations supplémentaires sur l'écriture de scripts

Vous avez à présent suffisamment de connaissances pour créer des scripts simples qui exécutent des tâches de base. Pour approfondir vos connaissances dans le domaine de l'écriture de scripts, reportez-vous aux documents suivants :

- « Techniques d'écriture de scripts avancées », page 46
- Le guide d'écriture de scripts Adobe de votre application.
- Chapitre 6, « Bibliographie »

3 Recherche de propriétés et de méthodes d'objets

Adobe fournit les ressources suivantes pour vous permettre de trouver et d'utiliser les objets, méthodes ou commandes, propriétés, énumérations et paramètres dont vous avez besoin pour créer des scripts efficaces.

- Dictionnaires d'objets ou bibliothèques de types. Chaque application Adobe compatible avec les scripts fournit une bibliothèque ou un dictionnaire de référence dans votre environnement d'éditeur de scripts.
- Les documents de référence sur l'écriture de scripts d'Adobe (au format PDF) qui se trouvent sur votre CD d'installation. (Les documents de référence sur l'écriture de scripts ne sont pas fournis avec toutes les applications Adobe.)

Utilisation de navigateurs dans un environnement de script

Cette section explique comment afficher et utiliser des navigateurs d'objets dans un environnement de script pour chaque langage de script.

Dictionnaires de données AppleScript

Les dictionnaires AppleScript sont disponibles dans l'application Editeur de scripts d'Apple.

Affichage des dictionnaires AppleScript

REMARQUE : L'emplacement par défaut de l'application Editeur de scripts est Applications > AppleScript > Editeur de scripts.

1. Dans l'Editeur de scripts, choisissez Fichier > Ouvrir le dictionnaire. L'Editeur de scripts affiche la boîte de dialogue Ouvrir un dictionnaire.
2. Choisissez votre application Adobe, puis cliquez sur Ouvrir. L'Editeur de scripts ouvre l'application Adobe et affiche ensuite le dictionnaire de l'application.

Utilisation des dictionnaires AppleScript

Le dictionnaire AS divise les objets en suites. Les noms de suites indiquent le type d'objets contenus dans la suite.

Pour consulter les propriétés d'un objet :

1. Dans la fenêtre en haut à gauche de l'écran du dictionnaire, sélectionnez la suite qui contient l'objet.
2. Sélectionnez l'objet dans la fenêtre de la partie haute de l'écran.

   REMARQUE: Les objets sont indiqués par une icône carrée :  ; les commandes le sont par une icône ronde : .

   La description d'un objet figure dans la fenêtre de visualisation inférieure. Les éléments et propriétés d'un objet sont énumérés sous sa description. Chaque nom d'élément constitue un lien hypertexte vers le type d'objet de l'élément.

3. Chaque liste de propriétés contient les informations suivantes :

   ▷ Le nom de la propriété.

   ▷ Le type de données entre parenthèses.

   Si le type de données est un objet, ce type de données constitue un lien hypertexte vers l'objet.

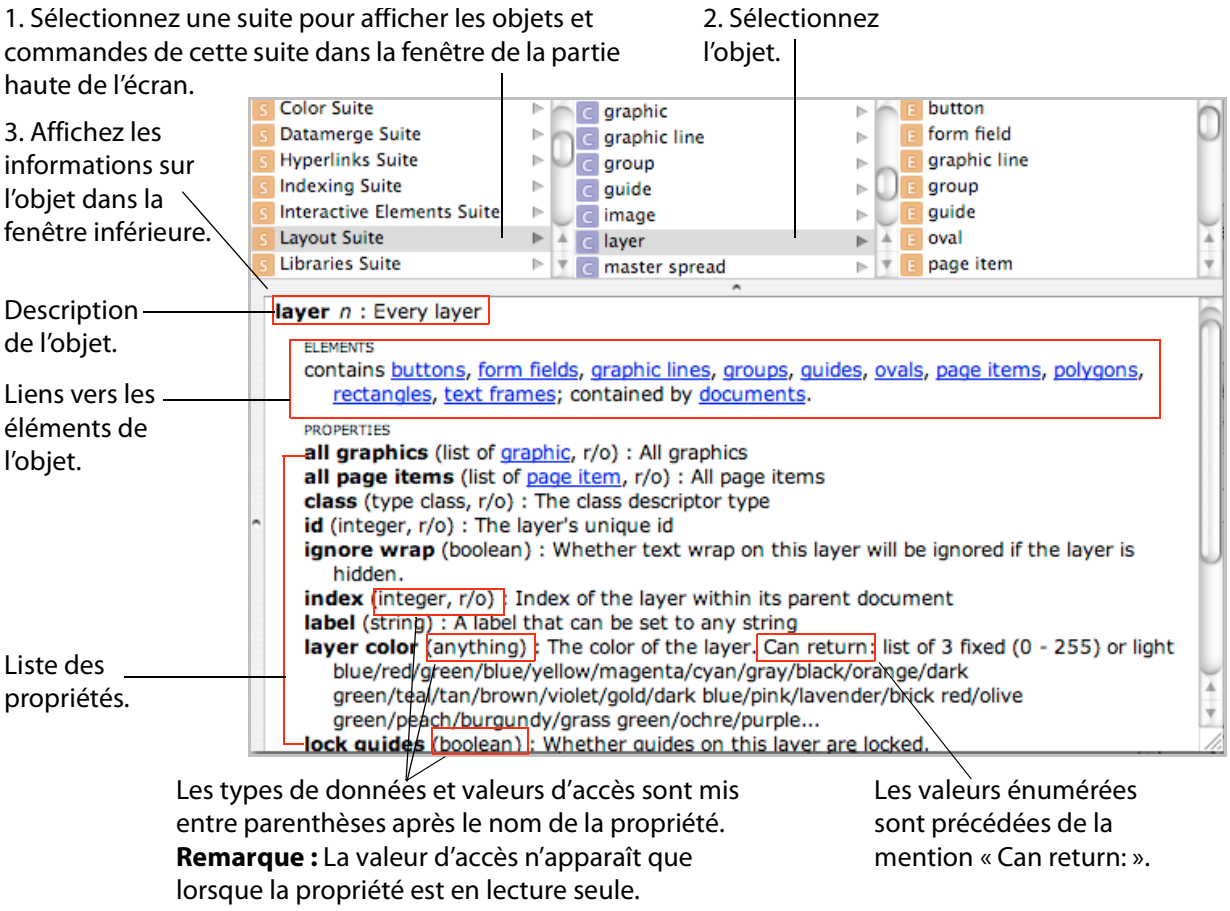
   Si le type de données est une énumération, le type de données est « anything » (n'importe lequel). Les valeurs valides sont listées après la description de la propriété et séparées par des barres obliques (/), elles sont également précédées de la mention « Can return: ».

   ▷ La valeur d'accès :

   Si l'objet est en lecture seule, `r/o` apparaît après le type de données.

   Si l'objet est en lecture-écriture, aucune valeur d'accès n'est donnée.

   ▷ Une description de la propriété.



Affichage des commandes et des paramètres des commandes

REMARQUE : Le dictionnaire de données liste les objets que vous pouvez utiliser avec une commande. Toutefois, il n'indique pas les commandes que vous pouvez utiliser avec un objet. Pour afficher une liste des commandes que vous pouvez utiliser avec un objet, reportez-vous à la référence d'écriture de scripts AppleScript de votre application. Reportez-vous à « Utilisation des documents de référence Adobe relatifs à l'écriture de scripts », page 41 pour plus d'informations.

Pour afficher les commandes dans le dictionnaire de données, suivez la procédure suivante :

1. Dans la fenêtre en haut à gauche de l'écran du dictionnaire de données, sélectionnez la suite qui contient la commande.

   La fenêtre de la partie haute de l'écran énumère les commandes et objets contenus dans la suite.

2. Sélectionnez la commande dans la fenêtre de la partie haute l'écran.

   REMARQUE : Les commandes sont indiquées par une icône ronde : ; les objets sont indiqués par une icône carrée : .

   La description d'une commande figure dans la fenêtre de visualisation inférieure.

   ▷ Sous la description, les objets avec lesquels vous pouvez utiliser la commande sont énumérés.

   ▷ Les paramètres sont énumérés sous la liste des objets pris en charge.

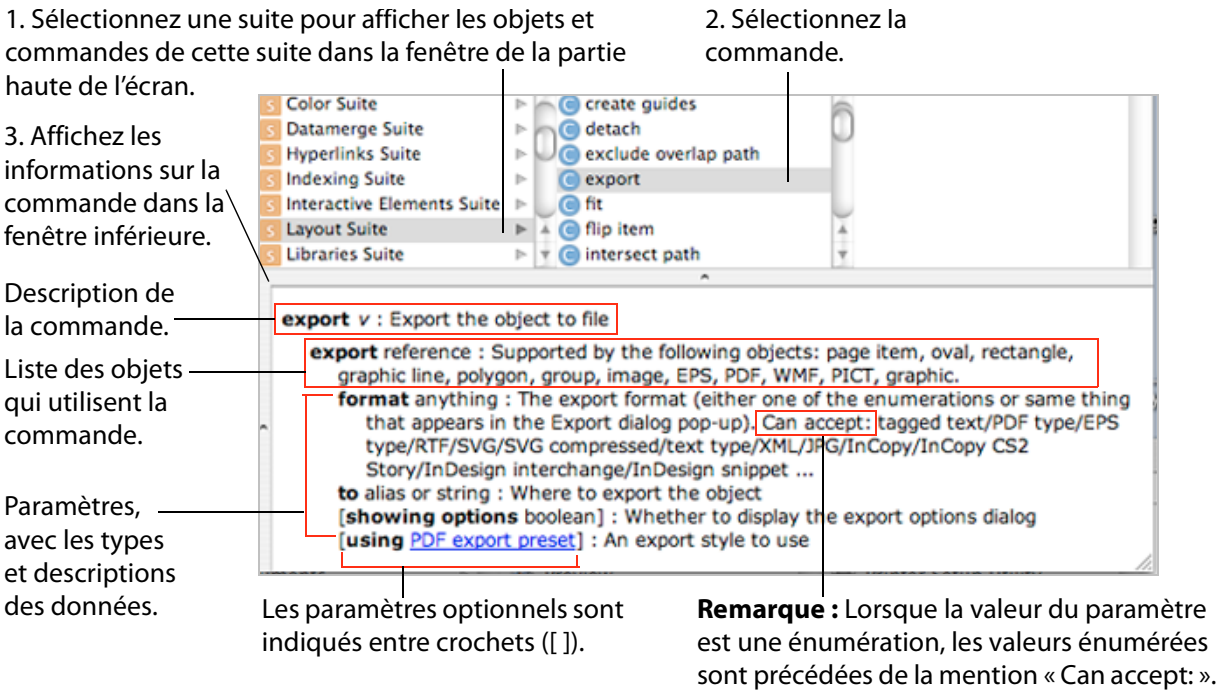
   Si le paramètre est optionnel, il est indiqué entre crochets ([ ]).

   Si aucun crochet n'apparaît avant et après le nom du paramètre, celui-ci est obligatoire.

   ▷ Chaque nom de paramètre est suivi du type de données.

   Si le type de données est un objet, ce type de données constitue un lien hypertexte vers l'objet.

   Si le type de données est une énumération, les valeurs valides sont précédées de la mention « Can return: », puis listées et séparées par des barres obliques (/).



Visionneuse de modèle objet (Object Model Viewer) JavaScript

Vous pouvez utiliser l'ESTK installé sur vos applications Adobe pour afficher les objets et méthodes JavaScript disponibles pour votre application Adobe.

Pour plus d'informations sur l'affichage et l'utilisation de la visionneuse de modèle objet JavaScript pour votre application Adobe, reportez-vous au *Guide JavaScript Tools* .

Bibliothèques de types VBScript

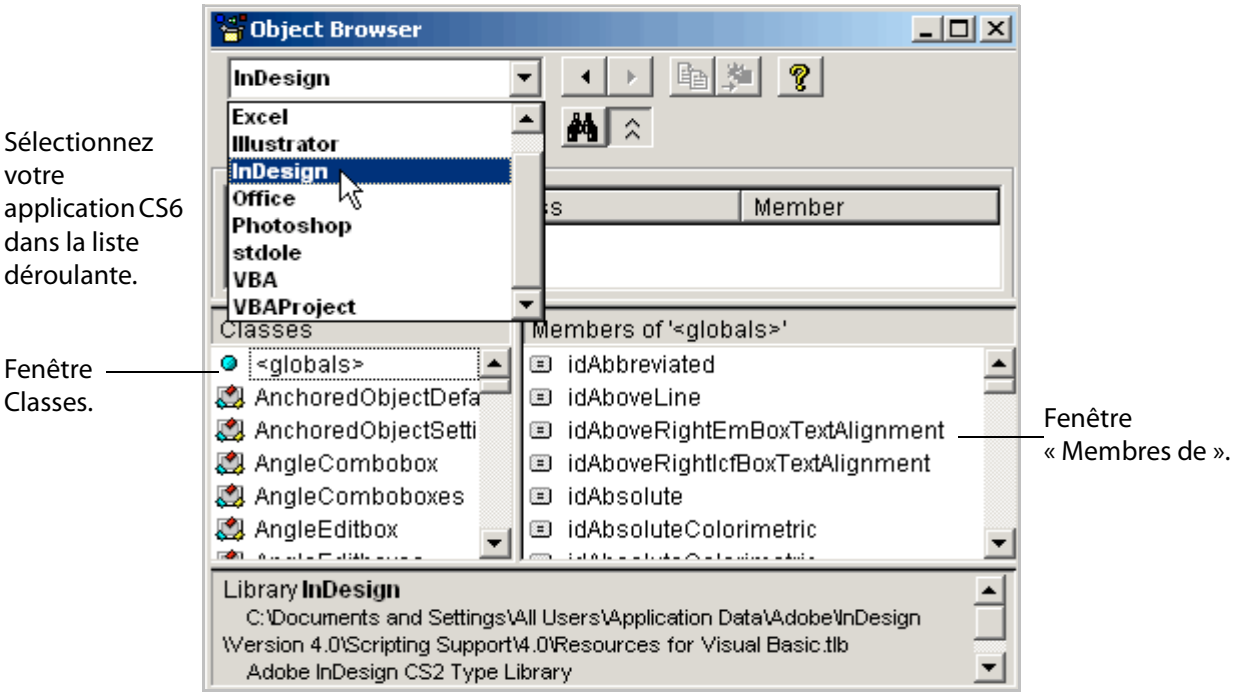
Vous pouvez utiliser l'éditeur Visual Basic (Visual Basic Editor) dans n'importe quelle application Microsoft Office pour afficher les objets et méthodes VBScript disponibles pour votre application Adobe.

REMARQUE : Si vous utilisez un éditeur différent, référez-vous à son système d'aide pour savoir comment afficher les bibliothèques de types.

Affichage des bibliothèques de types VBScript

Pour afficher la bibliothèque d'objets VBS, suivez la procédure suivante :

1. Démarrez une application Microsoft Office, puis sélectionnez Outils > Macro > Visual Basic Editor.
2. Dans la fenêtre Visual Basic Editor, choisissez Outils > Références.
3. Dans la liste Références disponibles de la boîte de dialogue Référence - Project, sélectionnez votre application Creative Suite, puis cliquez sur OK.
4. Dans la fenêtre Visual Basic Editor, sélectionnez Affichage > Explorateur d'objets.
5. Sélectionnez votre application Adobe dans la liste déroulante située dans le coin supérieur gauche de la fenêtre Explorateur d'objets.



Utilisation des bibliothèques de types VBScript

La bibliothèque de types d'objets VBS affiche les objets et les constantes dans la fenêtre Classes située à gauche dans la fenêtre Explorateur d'objets. Dans la fenêtre Classes :

- Les objets sont indiqués par l'icône suivante : 
- Les constantes sont indiquées par l'icône suivante : 

Pour afficher les propriétés et méthodes d'un objet, sélectionnez le type d'objet dans la fenêtre Classes. Les propriétés et méthodes sont listées dans la fenêtre Membres de située à droite de la fenêtre Classes.

- Les propriétés sont indiquées par l'icône suivante : 
- Les méthodes sont indiquées par l'icône suivante : 

Utilisation de listes de propriétés dans l'Explorateur d'objets

Lorsque vous sélectionnez une propriété dans la fenêtre Membres de, les informations concernant cette propriété s'affichent dans la fenêtre d'informations au bas de la fenêtre Explorateur d'objets comme suit :

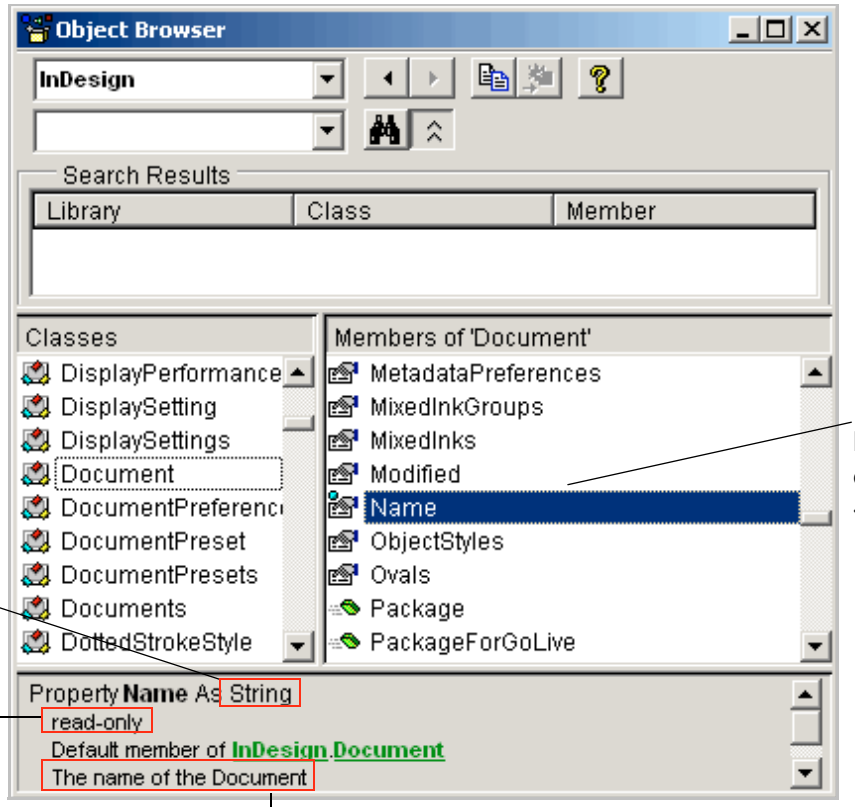
- Le nom de la propriété est suivi du type de données.
  - Si le type de données est une constante, cette constante constitue un lien hypertexte vers les valeurs de la constante. Les noms de constante commencent par un préfixe correspondant à l'abréviation du nom de l'application Adobe. Par exemple :

    Le préfixe `Ps` est utilisé pour des énumérations dans Photoshop CS6.
    Exemples : `PsColorProfileType`, `PsBitsPerChannelType`

    Le préfixe `id` est utilisé pour des énumérations dans InDesign CS6.
    Exemples : `idRenderingIntent`, `idPageOrientation`

    Le préfixe `Ai` est utilisé pour les énumérations dans Adobe Illustrator CS6.
    Exemples : `AiCropOptions`, `AiBlendModes`
  - Si le type de données est un objet, le nom de l'objet constitue un lien hypertexte vers le type d'objet.

- La valeur d'accès n'apparaît que lorsque la propriété est en lecture seule. Si la propriété est en lecture-écriture, aucune valeur d'accès n'apparaît.



1. Sélectionnez la propriété dans la fenêtre « Membres de ».

Le type de données apparaît à côté du nom de la propriété.

La valeur d'accès n'est répertoriée que lorsque l'accès est en lecture seule.

La description de la propriété figure au bas de la fenêtre d'informations.

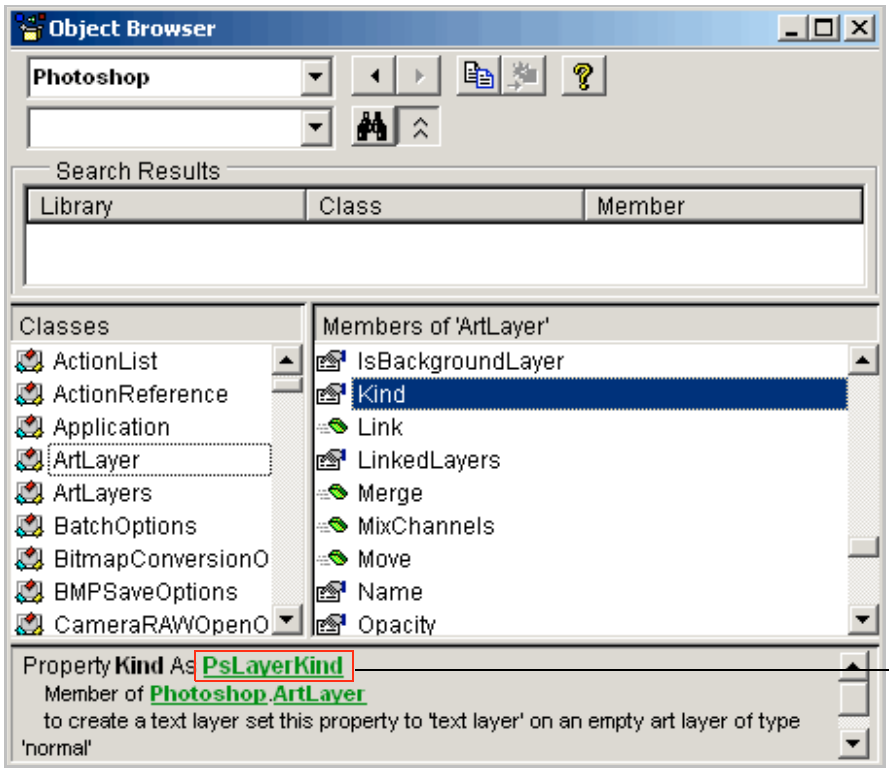
Recherche de la valeur numérique d'une énumération

En langage VBS, la valeur numérique d'une énumération est utilisée comme une valeur de propriété. Par exemple, dans le script ci-dessous, le type de calque, représenté par la propriété `Kind` dans la dernière ligne du script, est défini par la valeur numérique `2`, qui représente la valeur de la constante `TextLayer`.

```
Set appRef = CreateObject("Photoshop.Application")
Set docRef = appRef.Documents.Add
Set layerRef = docRef.ArtLayers.Add
    layerRef.Kind = 2 'PsTextLayer
```

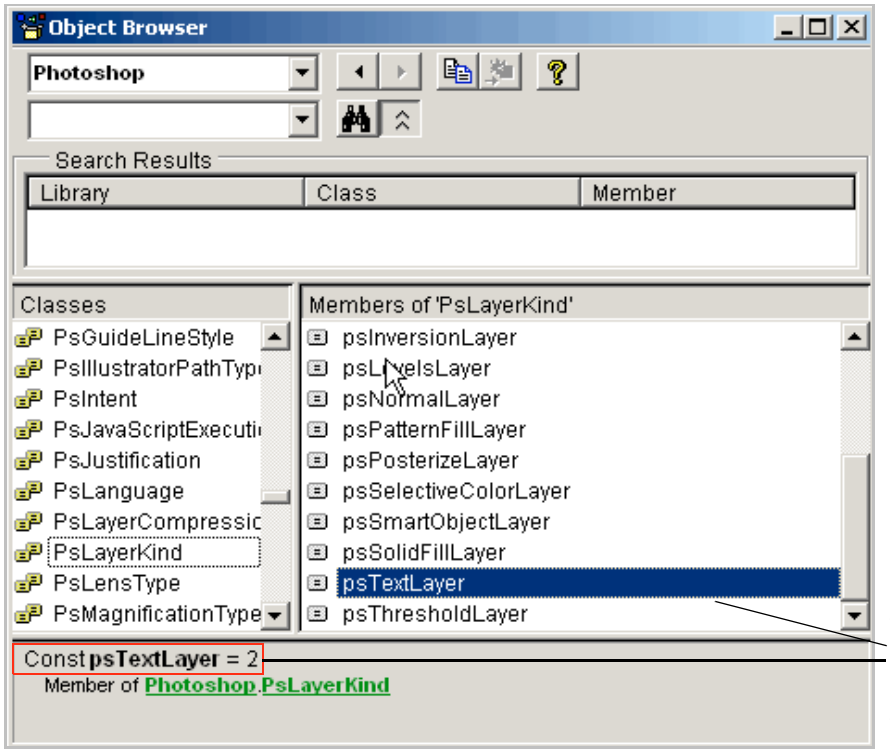
Pour rechercher la valeur numérique d'une énumération, suivez la procédure suivante :

1. Cliquez sur le lien menant aux informations concernant l'énumération.



Cliquez sur le lien menant aux informations concernant l'énumération.

2. Cliquez sur la valeur d'énumération pour afficher la valeur numérique dans la fenêtre du bas.



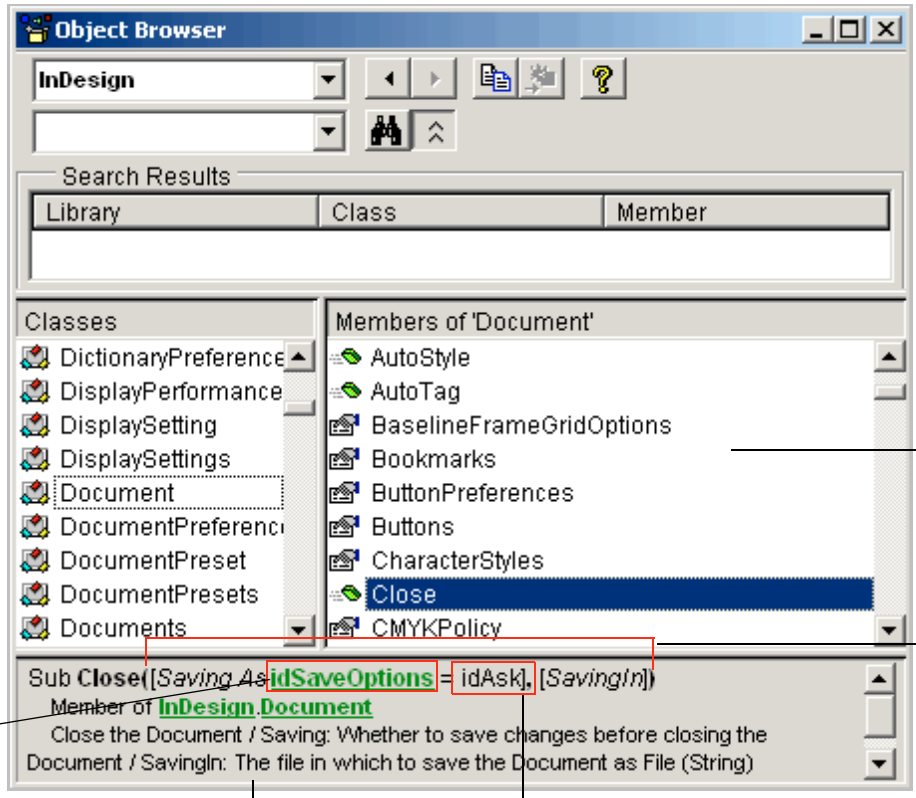
Cliquez sur la valeur d'énumération dans la fenêtre de droite pour afficher sa valeur numérique dans la fenêtre du bas.

Utilisation des listes de méthodes

Lorsque vous sélectionnez une méthode dans la fenêtre « Membres de », les informations concernant cette méthode s'affichent dans le volet d'informations au bas de la fenêtre Explorateur d'objets comme suit :

- Le nom de la méthode est suivi des paramètres.
  - Les paramètres optionnels sont indiqués entre crochets ([ ]).
  - Si aucun crochet n'apparaît avant et après le nom du paramètre, celui-ci est obligatoire.
- Chaque nom de paramètre est suivi du type de données.
  - Si le type de données est un objet, ce type de données constitue un lien hypertexte vers l'objet.
  - Si le type de données est une énumération, le nom de l'énumération commence par les initiales de l'application et constitue un lien hypertexte vers les informations concernant l'énumération.
  - Si une valeur par défaut existe pour un paramètre, cette valeur est listée après le type de données, après le symbole égal (=).

REMARQUE : La valeur par défaut est utilisée si vous ne définissez aucune valeur pour un paramètre. Seuls les paramètres optionnels ont des valeurs par défaut.



Le type de données est indiqué après le nom de la méthode. Si le type de données est une énumération, le nom de l'énumération commence par les initiales de l'application et constitue un lien hypertexte vers les informations concernant l'énumération.

La description de la méthode apparaît au bas de la fenêtre d'informations.

S'il existe une valeur par défaut, celle-ci est précédée du symbole égal (=).
Remarque : N'importe quel type de données peut avoir une valeur par défaut.

1. Sélectionnez la méthode dans la fenêtre « Membres de ».

Les paramètres sont répertoriés entre parenthèses après le nom de la méthode, les paramètres optionnels étant indiqués entre crochets ([ ]).

Utilisation des documents de référence Adobe relatifs à l'écriture de scripts

Adobe fournit des références relatives à l'écriture de scripts pour de nombreuses applications. Les références figurent sur votre CD d'installation.

Dans les références relatives à l'écriture de scripts, chaque langage est documenté dans un chapitre différent. Dans chaque chapitre, les objets sont énumérés par ordre alphabétique. Les tableaux suivants sont fournis pour chaque objet :

- Eléments (AS uniquement)
- Propriétés
- Méthodes, commandes ou fonctions

De plus, la plupart des sections d'objets contiennent un exemple d'écriture de script utilisant l'objet ainsi que certaines de ses propriétés, méthodes et commandes. Vous pouvez vous servir de n'importe quel script comme exemple ou point de départ pour votre propre script, dans lequel vous pourrez changer des propriétés ou des méthodes.

Utilisation du tableau d'éléments d'un objet (AS uniquement)

Les éléments sont les ensembles d'objets contenus par un objet. Lorsqu'un objet contient des éléments, un tableau indique les différents moyens de se référer aux éléments. Pour ceux qui débutent dans l'écriture de scripts, la chose la plus importante à retenir concernant le tableau d'éléments est la colonne du nom ou de l'élément, qui vous indique quels objets se trouvent directement en dessous de l'objet dans la hiérarchie de confinement. Par exemple, le tableau d'éléments suivant est tiré d'un objet `document` dans InDesign.

Nom	Référence à par
character style	indice, nom, plage, relatif, satisfaire à un test, ID
layer	indice, nom, plage, relatif, satisfaire à un test, ID
story	indice, nom, plage, relatif, satisfaire à un test, ID

Les informations que vous pouvez tirer de ce tableau sont que, dans les objets document que vous créez pour cette application, vous pouvez créer des objets `character style`, `layer` et `story`.

Par exemple :

```
tell application "Adobe InDesign CS6"
    set myDoc to make document
    set myCharStyle to make character style in myDoc with properties {name:"Bold"}
    set myLayer to make layer in myDoc
    set myStory to make story in myDoc
end tell
```

L'instruction de script suivante produira une erreur car `stroke style` n'est pas un élément de l'objet `document` de cette application.

```
tell application "Adobe InDesign CS6"
    set myDoc to make document
    set myStrokeStyle to make stroke style in myDoc with properties {name:"Erratic"}
end tell
```

Utilisation du tableau de propriétés d'un objet

Le tableau de propriétés d'un objet contient les informations suivantes :

- Les propriétés que vous pouvez utiliser avec l'objet
- Le type de valeur pour chaque propriété

  Lorsque le type de valeur est une constante ou une énumération, la valeur est présentée soit comme une liste de valeurs valides, soit comme un lien hypertexte vers la liste de constantes.

  Lorsque le type de valeur est un autre objet, la valeur est présentée comme un lien hypertexte vers la liste d'objets.

- Le statut d'entrée de la propriété : *lecture seule* ou *lecture-écriture*

- Une description qui comprend les informations suivantes :
  - Une explication de ce que fait ou définit la propriété
  - Les plages de valeurs valides
  - Les dépendances à d'autres propriétés

L'exemple suivant de tableau de propriétés pour un objet `art layer` dans Photoshop contient des exemples de chaque type de données.

Propriété	Type de valeur	Ce que c'est
`bounds`	Tableau de 4 chiffres	Lecture seule. Un tableau de coordonnées qui décrivent le rectangle limitatif du calque au format [y1, x1, y2, x2].
`kind`	`LayerKind`	Lecture seule. Le type de calque.
`name`	`string`	Lecture-écriture. Le nom du calque.
`opacity`	`number (double)`	Lecture-écriture. L'opacité en pourcentage. (Plage : de 0.0 à 100.0)
`textItem`	Objet `TextItem`	Lecture seule. L'élément de texte qui est associé au calque. **REMARQUE :** Valide uniquement quand `kind = LayerKind.TEXT`. Voir `kind`.
`visible`	`Booléen`	Lecture-écriture. Si la valeur est true (vraie), le calque est visible.

Par exemple :

AS

```
tell application "Adobe Photoshop CS6"
    set myDoc to make document
    set myLayer to make art layer in myDoc
    set properties of myLayer to {kind:text layer, name:"Captions", opacity:45.5, ¬
        visible:true}
    set contents of text object in myLayer to "Photo Captions"
end tell
```

REMARQUE : Vous ne pouvez pas définir les limites du calque car la propriété `bounds` est en lecture seule.

JS

```
var myDoc = app.documents.add()
var myLayer = myDoc.artLayers.add()
    alert(myLayer.bounds) // can't set the property because it is read-only
    myLayer.kind = LayerKind.TEXT
    myLayer.name = "Captions"
    myLayer.opacity = 45.5 // can use a decimal point because the type is not integer
    myLayer.textItem.contents = "Day 1: John goes to school"
    //see the properties table for the textItem object to find the contents property
    myLayer.visible = true
```

VBS

```
Set appRef = CreateObject("Photoshop.Application")
Set docRef = appRef.Documents.Add
Set layerRef = docRef.Layers.Add
    msgBox(layerRef.Bounds) ' can?t set the property because it is read-only
    layerRef.Kind = 2
    layerRef.Name = "Captions"
    layerRef.Opacity = 45.5 // can use a decimal point because the type is not integer
    layerRef.TextItem.Contents = "Day 1: John goes to school"
    //see the Properties table for the TextItem object to find the Contents property
    layerRef.Visible = true
```

REMARQUE : En JS et VBS, les objets d'ensemble sont conservés dans les propriétés de l'objet qui les contient. Pour déterminer la hiérarchie de confinement d'un objet, vous devez localiser l'objet ou les objets qui utilisent l'objet d'ensemble de cet objet (c'est-à-dire, la forme plurielle de l'objet) en tant que propriété. Par exemple, `documents.layers` ou `layers.textFrames`.

Utilisation du tableau de méthodes d'un objet

Le tableau de méthodes d'un objet contient les informations suivantes :

- Les méthodes que vous pouvez utiliser avec l'objet
- Le(s) paramètre(s) de chaque méthode
  - Lorsque le type de paramètre est une constante ou un autre objet, la valeur est présentée comme un lien hypertexte vers la constante ou la liste d'objets. Dans l'exemple suivant de tableau de méthodes, les types de paramètres `NewDocumentMode` et `DocumentFill` sont des constantes.
  - Les paramètres peuvent être obligatoires ou optionnels. Les paramètres optionnels sont indiqués entre crochets (`[]`).
- Le(s) type(s) de valeurs de retour, qui est (sont) ce que produit la méthode

  Lorsque le retour est une constante ou un autre objet, la valeur est présentée comme un lien hypertexte vers la constante ou la liste d'objets. Dans l'exemple suivant de tableau de méthodes, la valeur de retour `Document` est un objet.
- Une description, qui définit ce que fait la méthode

L'exemple suivant de tableau de méthodes énumère les paramètres pour la méthode `add` pour un document Photoshop CS6.

Méthode	Type de paramètre	Retours	Ce que cela fait
`add`		Document	Ajoute un objet document.
`[width]`	`UnitValue`		
`[, height]`	`UnitValue`		(`pixelAspectRatio`
`[, resolution])`	`number (double)`		Plage : `0.10` à `10.00`)
`[, name]`	`string`		
`[, mode])`	`NewDocumentMode`		
`[, initialFill]`	`DocumentFill`		
`[, pixelAspectRatio])`	`number (double)`		

Dans le tableau précédent :

- Tous les paramètres sont optionnels, comme indiqué par la présence des crochets.
- Les paramètres `width` et `height` sont par défaut les unités de règle actuelles et, en conséquence, le type de données est donc `UnitValue`. En d'autres termes, si l'unité de la règle verticale est en pouces et que celle de la règle horizontale est en centimètres, l'instruction suivante crée un document large de 5 pouces et haut de 7 centimètres :

  AS : `make document with properties {width:5, height:7}`

  JS : `app.documents.add(5, 7)`

  VBS : `appRef.Documents.Add(5, 7)`

- `mode` et `initialFill` prennent des valeurs constantes.

Les instructions de script suivantes définissent des valeurs pour chaque paramètre énuméré dans l'exemple de tableau des méthodes.

AS
```
make documement with properties {width:5, height:7, resolution:72," name:"Diary",
mode:bitmap, initial fill:transparent, pixel aspect ratio: 4.7}
```

JS
```
app.documents.add(5, 7, 72, "Diary", NewDocumentMode.BITMAP,
DocumentFill.TRANSPARENT, 4.7)
```

VBS
```
appRef.Documents.Add(5, 7, 72, "Diary", 5, 3, 4.7 )
```

4 Techniques d'écriture de scripts avancées

La plupart des scripts ne procèdent pas de manière séquentielle du début jusqu'à la fin. Il arrive souvent que les scripts prennent des chemins différents selon les données obtenues à partir du document en cours, ou qu'ils répètent les commandes à multiples reprises. Les *structures de contrôle* sont des fonctionnalités de langage de script qui permettent à vos scripts d'effectuer de telles actions.

Instructions conditionnelles

instructions if

Si vous pouviez parler à votre application Adobe, vous souhaiteriez pouvoir lui dire : « si le document n'a qu'un calque, crée un deuxième calque ». Cela constitue un exemple d'*instruction conditionnelle*. Les instructions conditionnelles prennent des décisions ; elles donnent à vos scripts un moyen d'évaluer quelque chose, comme le nombre de calques, et d'agir en fonction du résultat. Si la condition est satisfaite, le script exécute alors l'action comprise dans l'instruction `if`. Si la condition n'est pas satisfaite, le script ignore l'action comprise dans l'instruction `if`.

Chacun des scripts suivants ouvre un document puis vérifie si le document comprend un calque unique. S'il n'existe qu'un seul calque, le script ajoute un calque et règle l'opacité de remplissage du nouveau calque sur 65 %.

AS Une instruction `if` en langage AS commence par le mot `if` suivi par la phrase de comparaison entre parenthèses, puis du mot `then`. Vous devez fermer l'instruction `if` par `end if`.

```
tell application "Adobe Photoshop CS6"
    --modify the hard-drive name at the beginning of the filepath, if needed
    set myFilepath to alias "c:Applications:Adobe Photoshop CS6:Samples:Ducky.tif"
    open myFilepath
    set myDoc to current document
    tell myDoc
        if (art layer count = 1) then
            set myLayer to make art layer with properties {fill opacity:65}
        end if
    end tell
end tell
```

REMARQUE : AS utilise un symbole égal unique (=) pour comparer les valeurs.

Fermez maintenant `Ducky.tif` et essayez à nouveau le script en changeant l'instruction `if` comme suit :

```
if (art layer count < 1) then
```

JS Une instruction `if` en langage JS commence par le mot `if` suivi par la phrase de comparaison entre parenthèses. Placez l'action de l'instruction `if` entre accolades ({}).

```
var myDoc = app.open(File("/c/Program Files/Adobe/Adobe Photoshop
CS6/Samples/Ducky.tif"));
if(myDoc.artLayers.length == 1){
    var myLayer = myDoc.artLayers.add()
    myLayer.fillOpacity = 65
    }
```

REMARQUE : JavaScript utilise un symbole égal double (==) pour comparer les valeurs, au lieu d'un symbole égal simple (=) utilisé pour affecter des valeurs aux propriétés ou aux variables.

Fermez maintenant `Ducky.tif` et essayez à nouveau le script en changeant l'instruction `if` comme suit :

```
if(myDoc.artLayers.length < 1){
```

VBS Une instruction `if` en langage VBS commence par le mot `if` suivi par la phrase de comparaison, puis du mot `then`. Vous devez fermer l'instruction `if` par `End If`.

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Open("/c/Program Files/Adobe/Adobe Photoshop
CS6/Samples/Ducky.tif")
    If myDoc.ArtLayers.Count = 1 Then
        Set myLayer = myDoc.ArtLayers.Add
        myLayer.FillOpacity = 65
    End If
```

REMARQUE : VBS utilise un symbole égal unique (=) à la fois pour comparer et affecter les valeurs.

Fermez maintenant `Ducky.tif` et essayez à nouveau le script en changeant l'instruction `if` comme suit :

```
If myDoc.ArtLayers.Count < 1 Then
```

Instructions if else

Vous pouvez parfois avoir besoin de formuler des demandes légèrement plus complexes, telles que : « si le document a un calque, régler l'opacité de remplissage du calque sur 50 %, mais si le document a plusieurs calques, régler l'opacité du calque actif sur 65 % ». Ce type de situation nécessite une instruction `if else`.

AS

```
tell application "Adobe Photoshop CS6"
    --modify the hard-drive name at the beginning of the filepath, if needed
    set myFilepath to alias "c:Applications:Adobe Photoshop CS6:Samples:Ducky.tif"
    open myFilepath
    set myDoc to current document
    tell myDoc
        if (count of art layers < 2) then
            set fill opacity of current layer to 50
        else
            set fill opacity of current layer to 65
        end if
    end tell
end tell
```

JS

```
var myDoc = app.open(File("/c/Program Files/Adobe/Adobe Photoshop
CS6/Samples/Ducky.tif"));
if(myDoc.artLayers.length < 2){
    myDoc.activeLayer.fillOpacity = 50
    }
else{
    myDoc.activeLayer.fillOpacity = 65
    }
```

VBS

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Open("/c/Program Files/Adobe/Adobe Photoshop
CS6/Samples/Ducky1.tif")
If myDoc.ArtLayers.Count < 2 Then
    myDoc.ActiveLayer.FillOpacity = 50
Else
    myDoc.ActiveLayer.FillOpacity = 65
End If
```

Boucles

Il se peut que vous vouliez que votre script trouve et modifie tous les objets d'un certain type. Par exemple, votre document peut comporter des calques visibles et des calques invisibles et vous aimeriez les rendre tous visibles. Vous voudriez que ce script s'applique à plusieurs documents, mais vos documents possèdent un nombre différent de calques.

Dans une situation comme celle-ci, une instruction `repeat` (AS) ou une boucle (JS et VBS) peut vous être utile. Une boucle « passe » dans un ensemble d'objets et exécute une action pour chaque objet.

Pour utiliser les scripts de cette section, ouvrez votre application Adobe et créez un document qui contient au moins neuf calques. Rendez certains calques visibles et cachez les autres calques. Sauvegardez le document, puis exécutez le script, en substituant le nom de votre application et celui de l'objet `layer` dans le DOM de votre application.

Le principe de base à l'origine de ces boucles consiste dans le fait que le script identifie le premier calque dans l'élément ou l'ensemble et règle la visibilité de ce calque sur `true`, puis identifie le calque suivant et répète l'action, et ainsi de suite jusqu'à ce qu'une action ait été exécutée pour chaque calque.

AS

```
tell application "Adobe Illustrator CS6"
    set myDoc to current document
    tell myDoc
        set myLayerCount to (count layers)
        set myCounter to 1
        repeat while myCounter <= (myLayerCount + 1)
            set myLayer to layer myCounter
            set myLayer with properties {visible:true}
            --the next statement increments the counter to get the next layer
            set myCounter to myCounter + 1
        end repeat
    end tell
end tell
```

Ce script utilise deux variables, `myLayerCount` et `myCounter` pour identifier un calque, puis incrémente le numéro du calque jusqu'à ce que tous les calques aient été identifiés dans le document.

JS

```
var myDoc = app.activeDocument
var myLayerCount = myDoc.layers.length
for(var myCounter = 0; myCounter < myLayerCount; myCounter++)
    {var myLayer = myDoc.layers[myCounter]
    myLayer.visible = true}
```

Ce script utilise une boucle `for`, qui constitue l'une des techniques les plus courantes en JavaScript. Tout comme l'AppleScript mentionné plus haut, ce script utilise deux variables, `myLayerCount` et `myCounter`, pour identifier un calque, puis incrémente le numéro du calque jusqu'à ce que tous les calques du document aient été identifiés. L'incrémentation a lieu dans la troisième instruction, dans l'instruction `for` : `myCounter++`. La syntaxe `++` ajoute 1 à la valeur actuelle, mais n'ajoute pas 1 tant que l'action de boucle n'a pas eu lieu.

En langage commun, la boucle `for` de ce script peut correspondre à :

1. Commencer la valeur de `myCounter` à 0.
2. Si la valeur de `myCounter` est inférieure à la valeur de `myLayerCount`, utiliser alors la valeur de `myCounter` comme indice pour le calque assigné à `myLayer`, et régler la visibilité de `myLayer` sur `true`.
3. Ajouter 1 à la valeur de `myCounter`, puis comparer la nouvelle valeur de `myCounter` à la valeur de `myLayerCount`.
4. Si `myCounter` est toujours inférieur à `myLayerCount`, utiliser alors la nouvelle valeur de `myCounter` comme indice de `myLayer` et régler la visibilité de `myLayer` sur `true`, puis ajouter 1 à la valeur de `myCounter`.
5. Répéter jusqu'à ce que `myCounter` ne soit plus inférieur à `myLayerCount`.

VBS

```
Set appRef = CreateObject("Illustrator.Application")
Set myDoc = appRef.ActiveDocument
For Each object in myDoc.Layers
    object.Visible = True
Next
```

La boucle `For Each Next` en langage VBScript dit simplement à l'application de définir la propriété `Visible` de chaque objet de l'ensemble `Layers` du document actif sur `True`. Vous remarquerez que l'ensemble est identifié par la hiérarchie de confinement des objets parents (dans ce cas, par la variable `myDoc`) suivie du nom de l'ensemble, qui est la forme plurielle du nom de l'objet (dans ce cas `Layers`).

REMARQUE : L'objet nommé dans la boucle peut être n'importe lequel. Le script fonctionne de la même manière si vous substituez `x` à `object` dans le script ci-dessous :

```
Set appRef = CreateObject("Illustrator.Application")
Set myDoc = appRef.ActiveDocument

For Each x in myDoc.Layers

   x.Visible = True

Next
```

Informations supplémentaires sur l'écriture de scripts

Chaque langage de script contient un grand nombre supplémentaire de mécanismes et de techniques pour renforcer vos scripts et les rendre plus complexes. Pour continuer à vous informer sur la manière d'écrire des scripts pour vos applications Adobe, reportez-vous au guide d'écriture de scripts d'Adobe de votre application. Reportez-vous également au Chapitre 6, « Bibliographie ».

5 Résolution des problèmes

Ce chapitre explique comment interpréter certains messages d'erreur de base pouvant survenir lors de l'exécution d'un script.

Mots réservés

Tout comme de nombreux autres éditeurs de scripts, Script Editor et l'ESTK mettent certains mots en surbrillance lorsque vous les tapez.

Par exemple, les valeurs Booléennes `true` et `false` sont toujours en surbrillance. D'autres exemples sont listés ci-dessous.

AS

```
tell
end
with
set
```

JS

```
var
if
else
with
```

VBS

```
Dim
Set
MsgBox
```

Ces mots en surbrillance sont réservés par le langage de script à des fins spéciales et ne peuvent pas être utilisés comme noms de variable. Vous pouvez utiliser des mots réservés dans une chaîne, car ils seront alors entre guillemets. Vous pouvez également les utiliser dans des commentaires, car les commentaires sont ignorés par le moteur de script.

Si votre script indique une erreur de syntaxe, vérifiez que vous n'avez pas utilisé un mot réservé. Pour obtenir la liste complète des mots réservés dans votre langage de script, reportez-vous à l'une des ressources répertoriées au Chapitre 6, « Bibliographie ».

Messages d'erreur de l'éditeur de scripts AppleScript

Lorsque votre script AppleScript contient une erreur, l'Editeur de scripts met en surbrillance la partie erronée du script et affiche un message d'erreur.

Vérifiez la présence de fautes d'orthographe ou de ponctuation dans la partie du script mise en surbrillance. Si aucune erreur ne figure dans le texte en surbrillance, vérifiez le texte qui précède directement la partie en surbrillance. Si le texte qui précède contient une erreur, celle-ci peut avoir conduit l'éditeur de scripts à s'attendre à autre chose qu'à ce qui est indiqué dans la section mise en surbrillance.

Les messages d'erreur les plus courants sont expliqués ci-dessous.

- **Can't get *object*** : cela indique généralement que vous n'avez pas bien défini l'objet dans la hiérarchie de confinement. Essayez d'ajouter in `parent-object` (où `parent-object` est l'objet qui contient l'objet indiqué dans le message d'erreur) après le nom de l'objet dans votre script, ou créez une instruction imbriquée `tell` qui nomme l'objet parent.
- **Expected "" but found end of script** : assurez-vous que tous les guillemets ouvrent et ferment bien les chaînes.
- **Requested property not available for this object** : vérifiez l'orthographe des noms de toutes les propriétés.

CONSEIL : Sélectionnez le **journal des résultats (Result Log)** en bas de la fenêtre de l'éditeur de scripts pour voir la progression de votre script ligne par ligne.

Messages d'erreur de l'ESTK

L'ESTK vous prévient de la présence d'erreurs de plusieurs manières :

- Si votre script contient une erreur de syntaxe, il ne s'exécute pas et la section erronée est grisée. Une description du problème est souvent affichée dans la barre d'état au bas de la fenêtre ESTK.

  Lorsqu'une erreur de syntaxe a lieu, vérifiez les points suivants :

  - Assurez-vous d'avoir fait un usage correct des majuscules et des minuscules. Rappelez-vous que tous les termes en JavaScript (sauf les noms d'énumération) commencent par une minuscule, une majuscule étant utilisée au début de chaque mot d'un terme composé, comme ici : `artLayer`.

    Rappelez-vous également que les noms de variables sont sensibles à la casse.

  - Fermez toutes les parenthèses, toutes accolades et tous les guillemets. Assurez-vous que ces caractères vont bien par deux.

  - Assurez-vous d'utiliser des guillemets droits. Ne mélangez pas non plus les guillemets simples et doubles. Par exemple :

    Incorrect : `myDoc.name = "Mon document'`

    Correct : `myDoc.name = 'Mon document'`

    Correct : `myDoc.name = "Mon document"`

  REMARQUE : Certaines erreurs de syntaxe, telles que les guillemets droits ou courbes sont mises en surbrillance rouge. Le message de barre d'état dit simplement Syntax error (Erreur de syntaxe). Assurez-vous de bien utiliser des guillemets droits.

- Si votre script contient une erreur d'exécution, telle qu'un objet incorrectement identifié ou une propriété qui n'existe pas pour l'objet qu'elle essaye d'utiliser, l'instruction erronée est mise en surbrillance mais le script continue de s'exécuter, comme l'indique l'icône tournante dans le coin inférieur droit de l'écran. En outre, l'erreur est décrite à la fois dans la fenêtre de la console JavaScript et dans la barre d'état.

Lorsqu'une erreur d'exécution se produit :

- Sélectionnez **Debug (Débogage) > Stop (Arrêter)** ou appuyez sur la touche **Shift+F5** pour arrêter le script.
- Cherchez la nature de l'erreur dans la console JavaScript. De brèves descriptions des messages d'erreur les plus fréquents figurent ci-dessous et peuvent vous aider à savoir par où commencer.

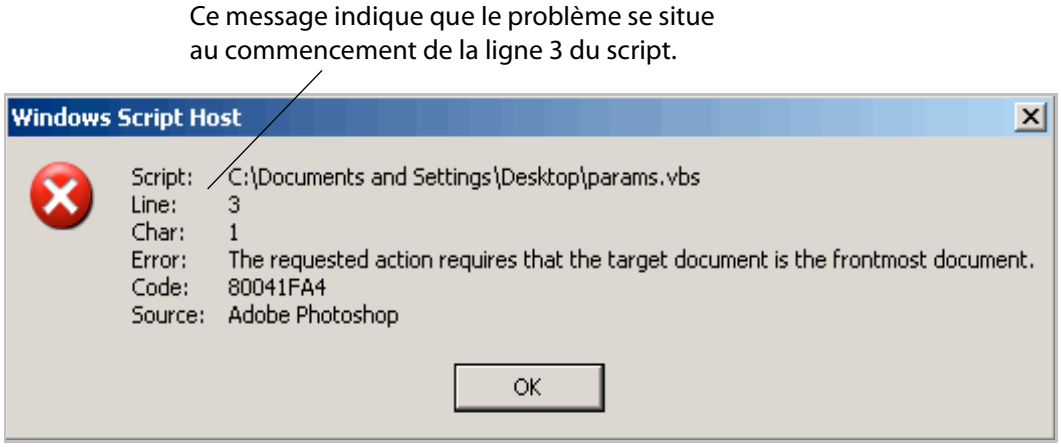
  element is undefined : si l'élément non défini est une variable, assurez-vous que le nom de cette variable ne contient pas de faute d'orthographe et que la casse est respectée. Assurez-vous également que la variable a été définie par une instruction `var` ou qu'une valeur lui a été attribuée.

  Si l'élément non défini est une valeur de chaîne, assurez-vous que cette valeur figure bien entre guillemets.

  undefined is not an object : assurez-vous que l'objet figurant dans l'instruction mise en surbrillance est correctement identifié dans la hiérarchie de confinement. Par exemple, si l'objet est un calque, assurez-vous d'avoir bien défini le document qui contient ce calque. Pour les objets document, il peut être nécessaire d'inclure l'objet parent `app`.

Messages d'erreur de VBScript

Lorsque votre script VBScript contient une erreur, un hôte de script Windows (Windows Script Host) affiche un message d'erreur qui indique à quelle ligne s'est produite l'erreur et à quel endroit de la ligne commence la syntaxe ou l'objet erroné.



6 Bibliographie

Ce chapitre contient une liste d'ouvrages sur l'écriture de scripts destinés aux débutants. Cette liste n'est pas exhaustive. Vous pouvez également rechercher des didacticiels en ligne pour votre langage de script.

AppleScript

Pour obtenir plus d'informations et d'instructions sur l'utilisation du langage de script AppleScript, reportez-vous aux documents et sources suivants :

- *AppleScript for the Internet: Visual QuickStart Guide*, 1st ed., Ethan Wilde, Peachpit Press, 1998. ISBN 0-201-35359-8.
- *AppleScript Language Guide: English Dialect*, 1st ed., Apple Computer, Inc., Addison-Wesley Publishing Co., 1993. ISBN 0-201-40735-3.
- *Danny Goodman's AppleScript Handbook*, 2nd ed., Danny Goodman, iUniverse, 1998. ISBN 0-966-55141-9.
- Site web AppleScript d'Apple Computer, Inc. :

  www.apple.com/applescript

JavaScript

Pour obtenir plus d'informations et d'instructions sur l'utilisation du langage de script JavaScript, reportez-vous aux documents et sources suivants :

- *JavaScript: The Definitive Guide*, David Flanagan, O'Reilly Media Inc, 2002. ISBN 0-596-00048-0
- *JavaScript Bible*, Danny Goodman, Hungry Minds Inc, 2001. ISBN 0-7645-4718-6
- *Adobe Scripting*, Chandler McWilliams, Wiley Publishing, Inc., 2003. ISBN 0-7645-2455-0

VBScript :

Pour obtenir plus d'informations et d'instructions sur l'utilisation du langage de script VBScript, reportez-vous aux documents et sources suivants :

- *Learn to Program with VBScript 6*, 1st ed., John Smiley, Active Path, 1998. ISBN 1-902-74500-0
- *Microsoft VBScript 6.0 Professional*, 1st ed., Michael Halvorson, Microsoft Press, 1998. ISBN 1-572-31809-0.
- *VBS & VBSA in a Nutshell*, 1st ed., Paul Lomax, O'Reilly, 1998. ISBN 1-56592-358-8
- Site web de scripts du réseau des développeurs Microsoft (MSDN) :

  msdn.microsoft.com/scripting

Index

J

L

M

O

P

S

T

V