

ADOBE® スクリプティング入門

© Copyright 2012 Adobe Systems Incorporated. All rights reserved.

Adobe® スクリプティング入門

NOTICE: All information contained herein is the property of Adobe Systems Incorporated. No part of this publication (whether in hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Adobe Systems Incorporated. The software described in this document is furnished under license and may only be used or copied in accordance with the terms of such license.

This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and non-infringement of third-party rights.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Creative Suite, Illustrator, InDesign, and Photoshop are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Apple, Mac OS, and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. Microsoft, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries. JavaScript and all Java-related marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. UNIX® is a registered trademark of The Open Group.

All other trademarks are the property of their respective owners.

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

目次

1	はじめに	5
	簡単に学習できるスクリプティング	5
	スクリプティングを使用する理由	5
	スクリプティングを導入するタイミング	5
	アクションまたはマクロについて	5
	実際のスクリプティングについて	6
	AppleScript	6
	VBScript	6
	JavaScript	7
	スクリプトの開始方法	8
	AS	8
	JS	8
	VBS	8
2	スクリプティングの基本	9
	スクリプティングの構成要素	9
	オブジェクト、プロパティ、メソッド、およびコマンドの概要	9
	オブジェクトの使用	9
	DOM のコンセプト	10
	変数	11
	オブジェクト参照のメリット	11
	変数をショートカットとして使用	12
	変数の命名	12
	オブジェクト参照としてのオブジェクトコレクションまたはオブジェクト要素	13
	要素とコレクションの後続アイテムに番号を付ける方法	14
	現在のオブジェクトまたはアクティブなオブジェクトの参照	15
	プロパティの使用	16
	AS	17
	JS	18
	VBS	19
	読み取り専用のプロパティと読み書きプロパティの概要	19
	アラートボックスを使用したプロパティの値の表示	19
	定数値と列挙	21
	AS	21
	JS	21
	VBS	22
	変数のプロパティ値への使用	22
	メソッドまたはコマンドの使用	23
	コマンドまたはメソッドのパラメーター	24
	必須パラメーター	24
	複数のパラメーター	25
	Tell 文 (AS のみ)	26

変数に関する注意	27
変数の値の変更	27
既存のオブジェクトを参照するための変数の使用	28
スクリプトファイルを読み易くする方法	28
スクリプトのコメント記述	28
AppleScript と VBScript での長い行の継続	29
配列の使用	30
オブジェクトの作成	30
スクリプトに関する追加情報	31
3 オブジェクトのプロパティおよびメソッドの検索	32
スクリプティング環境ブラウザーの使用	32
AppleScript データディクショナリ	32
AppleScript ディクショナリの表示	32
AppleScript (AS) ディクショナリの使用	32
JavaScript オブジェクトモデルビューア	35
VBScript タイプライブラリ	35
VBScript タイプライブラリの表示	35
VBScript タイプライブラリの使用	36
Adobe スクリプティングリファレンスの使用	40
オブジェクトの要素表の操作 (AS のみ)	40
オブジェクトのプロパティ表の操作	41
オブジェクトのメソッド表の操作	42
4 高度なスクリプティング技法	44
条件文	44
if 文	44
if else 文	45
ループ	46
スクリプトに関する追加情報	47
5 トラブルシューティング	48
予約語	48
AppleScript スクリプトエディターのエラーメッセージ	49
ESTK のエラーメッセージ	49
VBScript のエラーメッセージ	50
6 参考資料	51
AppleScript	51
JavaScript	51
VBScript	51
索引	52

1 はじめに

スクリプティングは、複数の Adobe® Creative Suite® アプリケーションの多くの機能を制御し自動化できる強力なツールです。時間や労力を大幅に軽減し、仕事に対するアプローチ方法を完全に変えることができます。

簡単に学習できるスクリプティング

スクリプティングはプログラミングではありません。一般的なさまざまなタスクを自動化する基本的なスクリプトを書くために、高度なコンピュータの知識は必要ありません。

各スクリプティングアイテムは、Adobe アプリケーションのツール、パレット、またはメニューアイテムに対応しています。つまり各スクリプティングの要素は、アドビ システムズ社の専門知識を通して既に理解している内容です。Adobe アプリケーションに何をさせたいのかがわかっているならば、簡単にスクリプトの書き方を学習できます。

スクリプティングを使用する理由

スクリプティングといえば創造性が特徴の分野ですが、その実際のタスクの多くは創造性とは程遠いものです。ほとんどの場合、同じか似たような手順を何度も繰り返して時間を浪費しています。

退屈でつまらないタスクを喜んで引き受け、完璧で予測できる一貫性を持って指示に従い、必要なときはいつでも利用可能で、迅速に仕事をこなし、絶対に請求書を送付してこないアシスタントがいたら便利だと思いませんか。

スクリプティングならそのアシスタントの代わりになれます。時間を浪費する簡単ではあるが反復的なタスクのスクリプトを、わずかな時間で学習できます。始めるのは簡単ですが、最新のスクリプティング言語にはかなり高度な仕事を扱う奥深さがあります。スクリプティング能力が向上するに伴い、寝ている間に一晩中働いてくれるもっと複雑なスクリプトを作成することが可能になります。

スクリプティングを導入するタイミング

反復的なタスクにいら立ったことはありませんか？ そうだとすれば、スクリプトを学習する候補者に選ばれたも同然です。次に、以下の情報を簡単に洗い出します。

- ▶ タスクの実行に含まれる手順。
- ▶ タスクを行うために必要な条件。

タスクを手動で実行する手順を理解していれば、スクリプトに変換する準備は完了です。

アクションまたはマクロについて

アクションを使用したこと、またはマクロを書いたことがあれば、スクリプトの効率のよさがわかります。スクリプティングはアクションまたはマクロを超えた機能を持ち、1つのスクリプトで複数のドキュメントや複数のアプリケーションを操作することができます。例えば、Adobe Photoshop® の画像を操作し、Adobe InDesign® に画像を取り込むように指示するスクリプトを書くことができます。

さらにスクリプトは、情報を取得したり応答する機能に極めて優れています。例えば、ドキュメントに異なるサイズの写真が含まれているとします。各写真のサイズを理解し、アイコンには青色の境界線、小さなイラストには緑色の境界線、半ページの画像には銀色の境界線というように、サイズに基づく異なった色の境界線を作成するスクリプトを書くことができます。

もしアクションを好んで使用しているのであれば、スクリプトはアプリケーション内でアクションを実行できることを覚えておいてください。

実際のスクリプティングについて

スクリプトとは、アプリケーションにタスクの実行を指示する一連の文です。

スクリプトの秘訣は、アプリケーションが理解できる言語で文を書くことです。Adobe アプリケーションは、複数のスクリプティング言語をサポートしています。

Mac OS[®] を使用している場合は、次から選択します。

▶ AppleScript

▶ JavaScript

Windows[®] を使用している場合は、次から選択します。

▶ VBScript (Visual Basic および VBA も使用可能)

▶ JavaScript

最適な言語を選択するには、次の説明を参考にします。

AppleScript

AppleScript は、Apple によって開発された「わかりやすい言葉」で書かれたスクリプティング言語です。最も簡単なスクリプティング言語の1つとされています。

AppleScript スクリプトを書くには、デフォルトの Mac OS インストールにある Apple のスクリプトエディターアプリケーションを使用します。デフォルトの Mac OS インストールは次の場所にあります。

システムドライブ / アプリケーション / AppleScript / スクリプトエディター

スクリプトエディターについて詳しくは、スクリプトエディターのヘルプを参照してください。

VBScript

VBScript は、Microsoft によって開発された Visual Basic プログラミング言語の縮小バージョンです。ActiveX Scripting を使用してホストアプリケーションと対話します。CS6 によって正式にサポートされている Visual Basic の言語バージョンは VBScript ですが、VBA および Visual Basic でスクリプトを書くことも可能です。

インターネット上で、質の良い VBScript エディターを見つけることができます。Microsoft Office アプリケーションがある場合は、**ツール／マクロ／ Visual Basic Editor** を選択して、内蔵されている Visual Basic Editor を使用することもできます。

JavaScript

JavaScript は、本来 Web ページを対話式に作成するために開発された一般的なスクリプティング言語です。AppleScript と同様、JavaScript は簡単に学習できます。

注意：アドビシステムズ社では、ExtendScript と呼ばれる JavaScript の拡張バージョンを開発しました。ExtendScript を使用することによって、特定の Adobe ツールおよびスクリプティング機能を利用することができます。初心者にとっては、これらの2つの言語の違いは問題にはなりません。ただし、JavaScript スクリプトには通常の .js 拡張子ではなく .jsx 拡張子をつける習慣をつけておく必要があります。

JavaScript には、AppleScript および Visual Basic に対して利点がいくつかあります。

- ▶ Windows または Mac OS の両方でスクリプトを使用することができます。スクリプトを両方のプラットフォームで共有または使用する場合は、JavaScript を学習する方が便利です。
- ▶ Adobe Illustrator[®] および InDesign では、アプリケーションからサポートされるどの言語でもスクリプトにアクセスできます。ただし、Photoshop では、アプリケーションから .jsx ファイルにしかアクセスできません。別のアプリケーションから AppleScript または Visual Basic スクリプトを実行する必要があります。これは重大な欠点ではありませんが、スクリプトを実行するために何度か追加的にマウスをクリックする必要があります。
- ▶ スクリプトをアプリケーションの Startup Scripts フォルダーに入れて、アプリケーションを開くと自動的に実行されるように .jsx スクリプトを設定できます。スクリプトフォルダーの起動について詳しくは、アプリケーションのスクリプティングガイドを参照してください。

JavaScript でスクリプトを書くには、お使いのテキストエディター、または Adobe アプリケーションに付属する ESTK (ExtendScript Tool Kit) を使用できます。ESTK には、スクリプトの問題点を特定し修正する方法を説明する内蔵型の構文チェッカー、ファイルを保存せずに ESTK からスクリプトを直接実行する機能など、テキストエディターよりも簡単に使用できる複数の機能があります。2 番目に説明した機能を使用すると、特にスクリプトをテストして動作するまで編集を繰り返す必要がある最初の段階で、時間を大幅に節約できます。

アドビシステムズ社のデフォルトインストールでは、ESTK は次の場所にあります。

Macintosh の場合：システムドライブ / アプリケーション / ユーティリティ / Adobe Utilities/ExtendScript Toolkit CS6

Windows の場合：システムドライブ：¥Program Files¥Adobe¥Adobe Utilities¥ExtendScript Toolkit CS6

詳しくは、『JavaScript ツールガイド』を参照してください。

スクリプトの開始方法

スクリプトを書いてみましょう。

注意：スクリプトの実行において問題が生じた場合は、[第5章「トラブルシューティング」](#)を参照してください。

AS

1. スクリプトエディターを開き、次のスクリプトを入力します（引用文の Adobe アプリケーション名は代替可能です）。

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

2. 「**実行**」をクリックします。

JS

1. ESTK を開き、ドキュメントウィンドウの左上隅のドロップダウンリストからアプリケーションを選択します。
2. JavaScript Console パレットで次を入力します。

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

3. 以下のいずれかの操作を行います。

- ▶ ドキュメントウィンドウ上部のツールバーにある実行アイコンをクリックします。
- ▶ **F5** キーを押します。
- ▶ **デバッグ／実行**を選択します。

VBS

1. テキストエディターで、次のスクリプトを入力します（2行目の引用文の Adobe アプリケーションは代替可能です）。

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

2. テキストファイルとして .vbs 拡張子でファイルを保存してください（例：create_doc.vbs）。
3. Windows エクスプローラーでファイルをダブルクリックします。

2 スクリプティングの基本

この章では、Windows OS と Mac OS におけるスクリプティングの基本コンセプトについて説明します。製品固有の手順については、Adobe アプリケーションのスクリプティングガイドを参照してください。

スクリプティングの構成要素

新しいドキュメントの作成に使用した最初のスクリプトは、英語の文章のように、名詞（document）と動詞（AS では make、JS では add()、および VBS では Add）で構成されました。スクリプティングでは、名詞はオブジェクト、動詞はコマンド（AS）、またはメソッド（JS と VBS）と呼ばれます。

形容詞を使用して名詞を修飾できるように、スクリプトオブジェクトもプロパティを使用して修飾できます。コマンドまたはメソッドを修飾するには、パラメーターを使用します。

オブジェクト、プロパティ、メソッド、およびコマンドの概要

Adobe アプリケーションを使用する場合、ファイルやドキュメントを開き、次にドキュメント内でレイヤー、テキスト、フレーム、チャンネル、グラフィックの線、色、およびその他のデザイン要素を作成または操作します。これらの要素がオブジェクトです。

スクリプト文を作成するには、オブジェクトを作成したり、既存のオブジェクトを参照したりしてから、次のいずれかの操作を行います。

- ▶ オブジェクトのプロパティの値を定義します。例えば、ドキュメントの名前、高さ、または幅を指定できます。さらに、レイヤーの名前、色、または不透明度を指定できます。
- ▶ オブジェクトに対して何を行うかをスクリプトに伝えるコマンドまたはメソッドを指定します。例えば、ドキュメントを開いたり、閉じたり、保存および印刷することができます。さらに、レイヤーを結合、移動、またはラスタライズできます。

スクリプトの作成時に注意することは、オブジェクトに許可されたプロパティかメソッド / コマンドのみを使用できることです。どのオブジェクトにどのプロパティとメソッドを使用できるかを確認するにはどうしたらいいでしょうか。たいていの場合、論理的に判断できます。通常、Adobe アプリケーションで指定できるものは、スクリプトでも指定できます。

しかし、アドビ システムズ社ではスクリプティングリソースでも詳細を解説しており、その中にはスクリプティングオブジェクトの作成、定義、および操作に必要な情報が含まれます。これらのリソースの場所と使用方法について詳しくは、[第3章「オブジェクトのプロパティおよびメソッドの検索」](#)を参照してください。

オブジェクトの使用

スクリプトでオブジェクトを使用する際に理解する必要のある主要コンセプトは、オブジェクトの参照方法です。スクリプトがどのオブジェクトを変更するかをアプリケーションに指示するにはどうしたらいいでしょうか。アプリケーションのユーザーインターフェイスでは、オブジェクトをクリックするだけで選択できます。スクリプトでは、手順が多少複雑になります。

DOM のコンセプト

スクリプティング言語では、ドキュメントオブジェクトモデル（DOM）と呼ばれるものを使用して、オブジェクトを特定しやすい方法で編成します。DOM には、包含階層という原理が使用されています。つまり、トップレベルオブジェクトには次のレベルのオブジェクトが包含され、この中にさらに下のレベルのオブジェクトが包含されます。

例えば、あらゆる Adobe アプリケーションの DOM におけるトップレベルオブジェクトは、アプリケーションオブジェクトです。その次はドキュメントオブジェクトで、この中にレイヤー、チャンネル、ページ、テキストフレームなど、他のすべてのオブジェクトが含まれます。これらのオブジェクトには、ドキュメントに直接含めることができないオブジェクトを含めることができます。例えば、InDesign か Illustrator では、テキストフレームに語句を含めることができます。ドキュメントには、テキストフレームがない限り、語句を含めることができません。同様に、Photoshop では、ドキュメントにレイヤーを含めることができ、レイヤーにはテキストフレームを含めることができますが、ドキュメントにレイヤーが含まれない場合は、テキストフレームを含めることができません。

注意：オブジェクトを含むオブジェクトは、親オブジェクトとも呼ばれます。

最初のスクリプトでは、最初にアプリケーションオブジェクトに名前をつけ（または、ESTK で名前を選択し）、次にそのアプリケーションでドキュメントを作成しました。次のステップとしてレイヤーを作成する場合は、スクリプトがレイヤーを作成するドキュメントを特定する必要があります。スクリプトがどこにオブジェクトを作成するかをアプリケーションにはっきりと指示しない場合、スクリプトは異常終了します。

注意：特定アプリケーション用の DOM 図については、アプリケーションのスクリプティングガイドを参照してください。

それでは、DOM の原理を使用して、ドキュメントにレイヤーを追加する方法について考えてみます（このスクリプトを Photoshop 用に修正する場合、レイヤーは AS では art layer、JS では artLayers、VBS では ArtLayers と呼ばれることに注意してください）。

```
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
```

これらのスクリプトを実行すると、アプリケーションはどのドキュメントが特定できないので、エラーが出ます。開いているのは1つのドキュメントしかないと思われるかもしれませんが、常にそうとは限りません。このため、スクリプティング言語は、すべてのスクリプト文ですべてのオブジェクトを明確に特定することを厳格に規定しています。

本マニュアルでは、オブジェクトを参照する3とおりの方法を説明します。

- ▶ 変数
- ▶ コレクションまたは要素の番号
- ▶ 「現在の」オブジェクトまたは「アクティブな」オブジェクトのプロパティ

変数

変数は、スクリプトにデータを保持するために作成するものです。このデータは変数の値と呼ばれ、スクリプトのオブジェクトか、オブジェクトを記述するプロパティとすることができます。変数は、オブジェクトまたは他のデータに付けるニックネームと考えることができます。

変数にオブジェクトを含める方法を使用すると、オブジェクトの参照が容易になります。ほとんどの人は、スクリプトに各オブジェクトに対する変数を作成します。

最初のスクリプトと同様、次のスクリプトはドキュメントを作成するものです。ただし、これらのスクリプトは、ドキュメントを含めるための myDoc という変数を作成します。これらのスクリプトを見て、最初のスクリプトと比較してください ([8 ページの「スクリプトの開始方法」](#)を参照してください)。

AS AS で変数を作成するには、set コマンドを使用し、その後に変数名を指定します。変数にデータ値を割り当てるには、to の後に値を指定します。

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

JS JS で変数を作成するには、var を使用し、その後に変数名を指定します。データ値を指定するには、等号 (=) の後に値を指定します。等号の前後にスペースがあるかどうかは問題になりません。

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

VBS VBS で変数を作成するには、Set コマンドを使用し、その後に変数名を指定します。データ値を指定するには、等号 (=) の後に値を指定します。等号の前後にスペースがあるかどうかは問題になりません。

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

オブジェクト参照のメリット

スクリプトで作成された document オブジェクトを参照する方法を用いると、レイヤーの追加が簡単になります (このスクリプトを Photoshop 用に修正する場合、レイヤーは AS では art layer、JS では artLayers、VBS では ArtLayers と呼ばれることに注意してください)。

AS

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

さらに有益なのは、レイヤーを保持する別の変数を作成できることです。これにより、レイヤーのプロパティを定義したり、レイヤーにオブジェクトを追加する場合に、レイヤーの参照が簡単になります。

```
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()
```

同じスクリプトですが、今回はレイヤーを保持する変数を作成します。

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

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

同じスクリプトですが、今回はレイヤーを保持する変数を作成します。

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

変数をショートカットとして使用

オブジェクトを保持する変数には、オブジェクトを特定する包含階層全体を含めることができます。例えば、`myLayer` を参照するのに、そのレイヤーを含むドキュメントを参照する必要はありません。次のスクリプトは、`myLayer` にテキストフレームを作成するものです。`myLayer` を使用する際には、レイヤーに関する包含階層情報を指定する必要はありません。

注意：次のスクリプトは、`contents` プロパティを使用してフレームにテキストを追加します。この時点では、`プロパティ`を使用するメカニズムを考える必要はありません。

次のスクリプトは、Illustrator CS6 オブジェクトモデルで定義されたオブジェクトとプロパティを使用しているため、例えば、InDesign や Photoshop では機能しません。

```
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!"
```

変数の命名

変数に記述的な名前を付けると、スクリプトがわかりやすくなります。`x` や `c` などの変数名は、スクリプトを再確認するときあまり意味を持ちません。`theDocument` や `myLayer` など、変数に含まれるデータを示す名前にした方が良いです。

変数名に標準的な接頭辞を使用すれば、スクリプティングシステムのオブジェクト、コマンド、およびキーワードから変数を区別しやすくなります。次に例を示します。

- ▶ Document オブジェクトを含む変数の先頭に「doc」接頭辞を使用したり（docRef など）、Art Layer オブジェクトを含む変数を識別するために「layer」接頭辞を使用します（layerRef および layerRef2 など）。
- ▶ 個人的な要素を追加するためには「my」接頭辞を使用して、スクリプトオブジェクトから個人的な変数を区別します。例えば、myDoc、myLayer、myTextFrame などが考えられます。

すべての変数名は、以下の規則に従う必要があります。

- ▶ 変数名は1語とする必要があります（スペースを含めない）。多くの人は各語の先頭を大文字にしたり（myFirstPage など）、アンダースコア文字（my_first_page）を使用してわかりやすい名前を付けています。変数名の先頭にはアンダースコア文字を使用できません。
- ▶ 変数名には数字を含めることができますが、数字で始めることはできません。
- ▶ 変数名には引用符や、アンダースコア文字を除く句読点を使用できません。
- ▶ JavaScript および VBScript の変数名は、大文字と小文字が区別されます。thisString は、thisstring や ThisString と同じではありません。AppleScript の変数名は、大文字と小文字が区別されません。
- ▶ スクリプトの各変数には、固有な名前を付ける必要があります。

オブジェクト参照としてのオブジェクトコレクションまたはオブジェクト要素

スクリプティング言語は各オブジェクトをコレクション（JS や VBS）または要素（AS）に収め、要素またはコレクション内でインデックスと呼ばれるオブジェクト番号を割り当てます。要素またはコレクション内のオブジェクトは、同一タイプのオブジェクトです。例えば、ドキュメント内の各 channel オブジェクトは channels 要素かコレクションに属し、各 art layer オブジェクトは art layers 要素か artLayers コレクションに属します。

英語では、「Give me the first document in the collection」（コレクションの最初のドキュメントを取得）と指定してドキュメントを参照することができます。スクリプティング言語でも、要素名かコレクション名とインデックスを使用して、同じようにオブジェクトを特定できます。

- ▶ AS では、documents 要素の最初のドキュメントを document 1 として参照します。
- ▶ JavaScript は、コレクションオブジェクトの番号を 0 から始めるので（これを覚えるのは最初は難しいものです）、JS の最初のドキュメントは documents[0] となります（インデックス前後の角かっこに注意してください）。
- ▶ VBS では、最初のドキュメントは Documents(0) となります（インデックス前後の丸かっこに注意してください）。VBS はコレクションオブジェクトの番号を 1 から始めます。

次のスクリプトは、新しいオブジェクトを追加するために、インデックスにより document オブジェクトと layer オブジェクトを参照しています。

注意：このスクリプトは変数を使用しないので、各オブジェクト参照に包含階層全体が必要になります。例えば、レイヤーを追加する文では、スクリプトがレイヤーを追加するドキュメントを特定する必要があります。レイヤーにテキストフレームを追加するには、スクリプトがフレームを含むレイヤーのインデックスを指定するだけでなく、レイヤーを含むドキュメントも特定する必要があります。

AS

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

注意：AppleScript を使用してスクリプトを作成した経験が少ないユーザーには、要素に複数のオブジェクトが含まれる場合に要素番号をオブジェクト参照として使用することは推奨できません。その理由について詳しくは、[14 ページの「要素とコレクションの後続アイテムに番号を付ける方法」](#)を参照してください。

JS

JavaScript では、アイテムのインデックスを示すのに、コレクション名の後にインデックスを角カッコ（[]）に入れて指定します。

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

注意：JS では、コレクションのインデックス番号が 0 から始まることに注意してください。

VBS

VBScript では、アイテムのインデックスを示すのに、コレクション名の後にインデックスを丸カッコに入れて指定します。

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

要素とコレクションの後続アイテムに番号を付ける方法

ここでは、2 番目のオブジェクトをコレクションまたは要素に追加したときに、スクリプティング言語が自動的に番号を付ける方法について解説します。

- ▶ AS は新しいオブジェクトに 1 番を割り当て、既に存在したオブジェクトに番号を付け直して 2 番とします。AppleScript オブジェクトの番号は、一番最後に使用したオブジェクトを示すようにオブジェクト内でシフトします。これにより、長いスクリプトでは混乱が発生しやすくなります。したがって、スクリプトを作成した経験があまりない人には、オブジェクト参照には変数を使用し、インデックスの使用を避けることが推奨されます。
- ▶ JS のコレクション番号は静的で、新しいオブジェクトをコレクションに追加しても変わることはありません。JS のオブジェクト番号は、オブジェクトがコレクションに追加された順番を示します。コレクションに最初に追加したオブジェクトには 0 番が割り当てられるので、次に追加したオブジェクトは 1 番になります。3 目のオブジェクトを追加すると、2 番になります。例えば、ドキュメントを追加すると、ドキュメントには自動的にレイヤーが含まれます。このレイヤーのインデックスは [0] です。レイヤーを追加すると、新しいレイヤーのインデックスは [1] になります。2 番目のレイヤーを追加すると、そのインデックスは [2] になります。レイヤー [2] をレイヤーパレットの一番下にドラッグしても、そのインデックスは [2] のままです。
- ▶ VBS のコレクション番号も変化することなく、VBS ではコレクションの最初のオブジェクトが常に (1) になることを除いて、JS のコレクションと同様に番号が付けられます。

ヒント：JS と VBS スクリプトでは、オブジェクト参照としてインデックス番号が非常に有益です。例えば、背景レイヤーを白にしたいファイルがいくつかある場合、「Open all files in this folder and change the first layer's color to white」（このフォルダーのすべてのファイルを開いて、最初のレイヤーの色を白に変更する）というスクリプトを作成できます。レイヤーをインデックスにより参照する機能がなければ、すべてのファイルのすべての背景レイヤーの名前をスクリプトに含めることが必要になります。

注意：スクリプトは強制的にオブジェクトをまとめます。コレクション全体にそのタイプのオブジェクトが 1 つしかない場合でも、スクリプトは要素またはコレクションの集合体にオブジェクトを配置します。

注意：オブジェクトは、複数のコレクションが要素に属することができます。例えば、Photoshop では、art layer オブジェクトは art layers 要素がコレクションに属し、layer set オブジェクトは layer sets 要素がコレクションに属しますが、art layer オブジェクトと layer set オブジェクトは共に layers 要素がコレクションに属します。これと同様に InDesign では、rectangle オブジェクトは rectangles 要素がコレクションに属し、text frame オブジェクトは text frames 要素がコレクションに属します。しかし、rectangle オブジェクトと text frame オブジェクトは共にページ上のあらゆるアイテムを含む page items 要素がコレクションに属します。この page items は楕円、グラフィックスの線、ポリゴン、ボタン、およびその他のアイテムを含みます。

現在のオブジェクトまたはアクティブなオブジェクトの参照

最初のスクリプトを実行し、新しいドキュメントを作成したときには、アプリケーションが開き、ドキュメントを作成しました。もしアプリケーションのユーザーインターフェイスでそのドキュメントを変更したければ、ドキュメントが自動的に選択されているので、そのままマウス、メニュー、ツールボックス、およびパレットで作業を行うことができます。

スクリプトで作成するすべてのオブジェクトでも、これは同様です。スクリプトで指定しない限り、新しいオブジェクトがアクティブなオブジェクトとなり、変更対象になります。

便利な機能として、多くの親オブジェクトに含まれるプロパティを使用して、簡単にアクティブなオブジェクトを参照することができます（プロパティについて詳しくは、本マニュアルで後述しています。ここでは各部を完全に理解せずに、このセクションのスクリプト文をコピーし、これがどのように機能するかを見るだけに留めてください）。

- ▶ AS では、アクティブなオブジェクトを参照するプロパティは、current という語とオブジェクト名で構成されます。この例をいくつか以下に示します。

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

- ▶ JS のプロパティ名は active とオブジェクト名を組み合わせた複合語です。JS での大文字と小文字の標準的な使用法は以下のとおりです。

▷ 複合用語の最初の単語は小文字です。

▷ 複合用語の 2 番目の単語（およびそれに続くすべての単語）は、先頭の文字が大文字になります。

この例をいくつか以下に示します。

```
activeDocument
activeLayer
activeChannel
activeView
```

- ▶ VBS は、複合用語のすべての単語の先頭が大文字になることを除いて、JS とまったく同じです。この例をいくつか以下に示します。

```
ActiveDocument
ActiveLayer
ActiveChannel
ActiveView
```

次のスクリプトは、ドキュメントを作成してから、この原則を使用して新しいドキュメントにレイヤーを作成します。

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

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

注意：activeDocument の後に s を付け不要に注意してください。

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

注意：ActiveDocument の後に s を付け不要に注意してください。

プロパティの使用

オブジェクトのプロパティを定義または変更するには、3つの手順に従います。

1. オブジェクトに名前を付けます。
2. プロパティに名前を付けます。
3. プロパティの値を指定します。

次のいずれかのデータ型を値に使用できます。

- ▶ テキストとして解釈される英数字テキストの文字列。文字列は引用符（"）で囲みます。文字列には、オブジェクト名などの値も含まれます。
- ▶ 加算や除算などの数値演算に使用できる数値。数値には、フレームの一方の長さや段落間のスペース、不透明度のパーセンテージ、フォントサイズ、線の幅などがあります。

数値のように見える値が、実際は文字列であることがあります。例えば、電話番号や社会保障番号は数字ですが、数値の数値とは見なされないため、文字列として書式設定します（引用符で囲む）。

数値カテゴリーの中には、さまざまなタイプの数字があります。

▷ 小数点の付かない整数。

▷ 5.9 や 1.0 などのように小数点を含めることのできる、real、fixed、short、long、double など。

注意：これらの違いは重要に見えないかもしれませんが、今後のために覚えておいてください。

- ▶ 変数。変数をプロパティ値に使用する場合は、変数を文字列のように引用符で囲みません。
- ▶ ブール値。true か false のいずれかです。

注意：ブール値は、オン / オフスイッチとして機能することがよくあります。

- ▶ 定数値（列挙とも呼ぶ）。既定の値のセットから選択できます。プロパティに定数値を使用するのは、Adobe アプリケーションのドロップダウンメニューを使用するのと概念的に同じです。定数とその使用方法および使用状況については、[21 ページの「定数値と列挙」](#)で説明します。

- ▶ リスト（AS）または配列（JS および VBS）。

プロパティの中には、点の位置のページ座標（x 座標と y 座標）、またはテキストフレームや幾何学的なオブジェクトの境界などのように、複数の値が必要なものがあります。単一のプロパティに複数の値がある場合、AS ではこれをリスト、JS または VBS では配列と呼びます。各言語で、書式設定ルールを指定します。

- ▷ リストや配列は、次のように囲む必要があります。

AS のリストは波かっこ：{ } で囲みます。

JS の配列は角かっこ：[] で囲みます。

VBS では、配列は丸かっこで囲み、Array:Array() キーワードを続けます。

- ▷ 値はコンマ（,）で区切ります。コンマの後にはスペースを含めても省略しても構いません。

AS： {3,4,5} または {"string1", "string2", "string3"}

JS： [3,4,5] または ["string1", "string2", "string3"]

VBS： Array(3,4,5) または Array("string1", "string2", "string3")

- ▷ リストや配列には、ページ座標のリストなど、ネストされたリストや配列を含めることもできます。次のサンプルでは、ネストされた各配列が個別に囲まれ、ネストされた配列はコンマで区切られていることに注意してください。

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

AS でプロパティを使用するには、set コマンドに続けてプロパティ名を入力し、of の後にオブジェクト参照を入力します。次のスクリプトは、layer オブジェクトの name プロパティを定義します。

```
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
```

properties プロパティを使用して、1 つの文に複数のプロパティを設定することができます。複数のプロパティは配列として書式設定し、波かっこで囲みます。配列内の各プロパティ名 / プロパティ値の対はコロン（:）で区切ります。次のスクリプトは properties を使用して、レイヤーの名前と表示状態を定義します。

```
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
```

注意：前述のスクリプトでは、"My New Layer" の文字列値だけが引用符で囲まれていることに注意してください。visible プロパティの値の false は文字列のように見えますが、これはブール値です。値のタイプを確認するには、[16 ページの「プロパティの使用」](#)を参照してください。

次のスクリプトのように、オブジェクトを作成する文でオブジェクトのプロパティを定義することができます。

```
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

JS でプロパティを使用するには、プロパティを定義する、または変更するオブジェクトの名前を入力し、ピリオド (.) を挿入してからプロパティ名を入力します。値を指定するには、プロパティ名の後に等号 (=) を配置してから、値を入力します。

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

複数のプロパティを定義するには、複数の文を作成します。

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

注意：前述のスクリプトでは、"My New Layer" の文字列値だけが引用符で囲まれていることに注意してください。visible プロパティの値の false は文字列のように見えますが、これはブール値です。値のタイプを確認するには、[16 ページの「プロパティの使用」](#)を参照してください。

JS には、with 文と呼ばれる、複数のプロパティを定義する省略表現があります。with 文を使用するには、with の単語の後に定義するプロパティを持つオブジェクトを続け、オブジェクト参照を丸かっこ (()) で囲みます。with と最初の丸かっこの間にスペースを入れないでください。次に左波かっこ ({) を入力し、**Enter** キーを押して、次の行にプロパティ名と値を入力します。with 文を閉じるには、右波かっこ (}) を入力します。

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

with 文を使用すると、各プロパティでオブジェクト参照とピリオド（この場合は myLayer.）を入力する手間が省けます。with 文を使用する場合は、常に右波かっこを忘れないようにしてください。

JS には、1 つの文で複数の値を定義できる properties プロパティもあります。値の全グループを波かっこで囲みます ({}). 波かっこ内でプロパティ名と値を区別する場合はコロン (:) を使用し、プロパティ名 / プロパティ値の対を区別する場合はコンマ (,) を使用します。

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

VBS

VBS でプロパティを使用するには、オブジェクト名を入力してピリオド (.) を挿入し、プロパティ名を入力します。値を指定するには、プロパティ名の後に等号 (=) を配置してから、値を入力します。

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

1 つの文につき 1 つのプロパティのみを定義できます。複数のプロパティを定義するには、複数の文を作成する必要があります。

```
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
```

注意：前述のスクリプトでは、"My New Layer" の文字列値だけが引用符で囲まれていることに注意してください。Visible プロパティの値の false は文字列のように見えますが、これはブール値です。値のタイプを確認するには、[16 ページの「プロパティの使用」](#)を参照してください。

読み取り専用のプロパティと読み書きプロパティの概要

プロパティ値を定義する場合、完全な構文のスクリプト文を作成したのに結果が生成されないことがあります。「書き込み可能」ではないプロパティ、つまり読み取り専用のプロパティを定義した場合に、この状況が起きることがあります。

例えば、ほとんどの Adobe アプリケーションのドキュメントオブジェクトの name プロパティは読み取り専用です。したがって、スクリプトを使用して既存のドキュメントの名前を定義したり変更したりすることはできません（しかし save as コマンド、またはメソッドを使用して定義または変更することはできます。詳しくは、[23 ページの「メソッドまたはコマンドの使用」](#)を参照してください）。設定できないプロパティを持つことに疑問があるかもしれません。しかし、読み取り専用プロパティは貴重な情報源です。例えば、ドキュメントの名前や Documents コレクションのドキュメント数などを知りたい場合に有益です。

アラートボックスを使用したプロパティの値の表示

読み取り専用プロパティの情報を表示するには、アラートボックスと呼ばれる情報を表示する小さいダイアログを使用すると良いです。アラートボックスを使用すると、読み書きプロパティでも読み取り専用プロパティでも、任意のプロパティの値を表示できます。

AS

AS でアラートボックスを表示するには、display dialog と入力してから、ダイアログの内容を丸かっこ (()) の中に入力します。要素内のオブジェクト数を検索するには、count コマンドを要素名と共に使用します。

注意：要素名には、オブジェクトの複数形が使用されます。例えば、document オブジェクトの要素は documents オブジェクトになります。

次のスクリプトは、documents 要素内のドキュメント数を表示し、ドキュメントを追加してから、更新された数と共に新しい警告を表示します。

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

アラートボックスに表示される文字列の値を取得するには、文字列を変数で保存する必要があります。次のスクリプトは、ドキュメント名を myName という変数に変換してから、myName の値を表示します。

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

JS

JS でアラートボックスを表示するには、alert と入力してからダイアログの内容を丸かっこ (()) に入力する alert() メソッドを使用します。alert と最初の丸かっこの間にスペースを入れないでください。コレクション内のオブジェクト数を検索するには、コレクションオブジェクトの length プロパティ（読み取り専用）を使用します。次のスクリプトは、documents コレクション内のドキュメント数を表示し、ドキュメントを追加してから、更新された数と共に新しい警告を表示します。

注意：コレクションオブジェクトには、オブジェクトの複数形が使用されます。例えば、document オブジェクトのコレクションオブジェクトは documents オブジェクトになります。

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

次のスクリプトは、アラートボックスにドキュメントの名前を表示します。

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

VBS

VBS でアラートボックスを表示するには、MsgBox と入力してからダイアログの内容を丸かっこ (()) に入力する MsgBox メソッドを使用します。MsgBox と最初の丸かっこの間にスペースを入れないでください。コレクション内のオブジェクト数を検索するには、コレクションオブジェクトの Count プロパティ（読み取り専用）を使用します。次のスクリプトは、Documents コレクション内のドキュメント数を表示し、ドキュメントを追加してから、更新された数と共に新しい警告を表示します。

注意：コレクションオブジェクトには、オブジェクトの複数形が使用されます。例えば、Document オブジェクトのコレクションオブジェクトは Documents オブジェクトになります。

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

次のスクリプトは、アラートボックスにドキュメントの名前を表示します。

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

定数値と列挙

プロパティの値の中には、アプリケーションで事前定義されているものがあります。例えば、ほとんどのアプリケーションのページの方法は、landscape か portrait です。これらの2つの値のいずれかしか使用できません。vertical、upright、horizontal、on its side などは使用できません。スクリプトで、ドキュメントのページ方向プロパティに使用できる値のみを設定するために、事前定義の値のみを持つプロパティが作成されます。

スクリプティングではこれらの事前定義の値を定数、または列挙と呼びます。

定数や列挙の使用方法は、アプリケーションのユーザーインターフェイスでドロップダウンリストを使用するのに似ています。

注意：プロパティの値に列挙を使用しなければならないかどうかを調べるには、アドビ システムズ社から提供されているスクリプティングリファレンスのプロパティを確認してください。詳しくは、[第3章「オブジェクトのプロパティおよびメソッドの検索」](#)を参照してください。

AS

AS では、他のプロパティ定義と同じように定数を使用します。定数は引用符で囲まないでください。次のスクリプトは、dark green の定数値を使用して、新しいレイヤーのレイヤーカラーを設定します。

```
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
```

注意：dark green が定数ではなく文字列値の場合は、この値を引用符で囲みます。

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

JS

JS では、列挙名、ピリオド (.)、列挙値の順に入力します。アドビ システムズ社から提供されているスクリプティングリファレンスで定義されている列挙と完全に一致するスペルや大文字と小文字を使用する必要があります。Adobe アプリケーションによって、書式が異なります。次に例を示します。

▶ InDesign の場合：

- ▷ 各列挙は大文字で始まり、組み合わせた用語内の単語もすべて大文字で始まります。
- ▷ 列挙値は小文字で始まります。

次の例は UIColor 列挙を使用して、レイヤーカラーを濃い緑に設定します。

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

var myLayer = mydoc.layers.add()

myLayer.layerColor = UIColor.darkGreen
```

▶ Illustrator の場合：

- ▷ 各列挙は大文字で始まり、組み合わせた用語内の単語もすべて大文字で始まります。
- ▷ 列挙値の中には、大文字で始まり、小文字を使用するものがあります。すべて大文字を使用するものもあります。スクリプティングリファレンスに記載されているものと完全に一致する値を使用する必要があります。

次の例は、RulerUnits 列挙を使用して、初期設定単位をセンチメートルに設定します。

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

myDoc.rulerUnits = RulerUnits.Centimeters
```

次のスクリプトは、値がすべて大文字で表される BlendModes 列挙を使用します。

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

var myLayer = myDoc.layers.add()

myLayer.blendingMode = BlendModes.COLORBURN
```

▶ Photoshop の場合：

- ▷ 各列挙は大文字で始まり、組み合わせた用語内の単語もすべて大文字で始まります。
- ▷ 列挙値はすべて大文字です。

次の例では LayerKind 列挙を使用して、レイヤーをテキストレイヤーにします。

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

myLayer.kind = LayerKind.TEXT
```

VBS

VBS では、定数に数値を使用します。

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

変数のプロパティ値への使用

プロパティの値を持った変数を使用することができます。これにより、スクリプトを迅速かつ正確に更新できます。例えば、すべての写真が 3 x 5 インチである出版物があるとしましょう。もし、写真の高さと幅をセットした変数を使用して、寸法が変更になった場合、ドキュメントの各写真の寸法を変更しなくても 1 つの変数の値を変更するだけで済みます。

次のスクリプトは、ドキュメントの幅と高さを持った変数を作成し、幅と高さを変更する文の値としてその変数を使用します。

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
```

注意: MsgBox メソッドは、一部の Adobe アプリケーションのスクリプトメニューからスクリプトを開いた場合には使用できません。メッセージボックスを正しく表示するには、Windows エクスプローラーでスクリプトファイルをダブルクリックします。

メソッドまたはコマンドの使用

コマンド (AS) とメソッド (VBS および JS) は、タスクを実行したり結果を取得するためにスクリプトに加える命令です。例えば、ドキュメントを印刷するには、`print/print()/PrintOut` コマンド/メソッドを使用します。

AS AS コマンドはスクリプトの文の先頭に命令動詞として表れます。コマンドの後には、このコマンドで操作するオブジェクトへの参照が続きます。

次のスクリプトはアクティブなドキュメントを印刷します。

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

JS メソッドは JS 文の終わりに挿入します。メソッド名の前にピリオドを入力してから、メソッド名を丸かっこ (()) で囲みます。

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

VBS メソッドは VBS 文の終わりに挿入します。メソッド名の前にピリオドを入力する必要があります。

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

コマンドまたはメソッドのパラメーター

コマンドやメソッドによっては、「引数」または「パラメーター」と呼ばれる追加のデータが必須です。パラメーターが付くことがオプションであるコマンドやメソッドもあります。

必須パラメーター

次のスクリプトは merge コマンドを使用しています。このコマンドでは、選択したレイヤーに結合するレイヤーを示す必要があります。プロパティと同様に、コマンドのパラメーターは波かっこ（{ }）で囲みます。ただし、波かっこ内にはパラメーターの値だけで、パラメーター名は含めません。

注意：このスクリプトは InDesign 用です。Illustrator には結合処理がありません（このスクリプトを Photoshop 用に修正する場合、レイヤーは AS では art layer、JS では artLayers、VBS では ArtLayers と呼ばれることに注意してください）。

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

メソッド名に続いてメソッドのパラメーターを丸かっこで囲みます。

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

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

myLayer2.merge(myLayer)
```

VBS

メソッド名の後のメソッドのパラメーターが丸かっこで囲まれています。最初の丸かっこの前にスペースを入力しないでください。

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

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

myLayer2.Merge(myLayer)
```

複数のパラメーター

コマンドやメソッドに複数のパラメーターを定義する場合は、次の規則に従ってください。

AS

AS コマンドには 2 種類のパラメーターがあります。

- ▶ ダイレクトパラメーターは、コマンドで実行するアクションの直接目的語を定義します。
- ▶ ラベル付きパラメーターは、ダイレクトパラメーター以外のすべてのパラメーターです。

ダイレクトパラメーターはコマンドの後に直接入力する必要があります。次の文では、コマンドは `make` で、ダイレクトパラメーターは `document` です。

```
make document
```

ラベル付きパラメーターは任意の順序で挿入できます。次のスクリプトでは 2 つのレイヤーを作成し、各レイヤーの場所と名前を定義します。レイヤーを作成する文で、`location` パラメーターと `name` パラメーターは異なる順序で表示されています。

```
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

JS では、どの値がどのパラメーターを定義するかをスクリプトコンパイラーが判別できるように、スクリプティングリファレンスリソースのリストの順にパラメーター値を入力する必要があります。

注意：スクリプティングリファレンスリソースについては、[第3章「オブジェクトのプロパティおよびメソッドの検索」](#)を参照してください。

オプションのパラメーターを省くには、プレースホルダー `undefined` を入力します。次の文は、幅が 4000 ピクセル、高さが 5000 ピクセル、解像度が 72、名前が「**My Document**」、ドキュメントモードがビットマップの Photoshop CS6 ドキュメントを作成します。

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

次の文は、解像度を未定義にして同じドキュメントを作成します。

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

注意：プレースホルダー `undefined` は、定義したいパラメーターに「達する」ためだけに使用します。次の文はドキュメントの高さと幅のみを定義しています。後に続くオプションのパラメーターにプレースホルダーは不要です。

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

VBS

VBS では、どの値がどのパラメーターを定義するかをスクリプトコンパイラーが判別できるように、リストの順にパラメーター値を入力する必要があります。

オプションのパラメーターを省くには、プレースホルダー `undefined` を入力します。次の文は、幅が 4000 ピクセル、高さが 5000 ピクセル、解像度が 72、名前が「My Document」、ドキュメントモードがビットマップの Photoshop CS6 ドキュメントを作成します。

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

次の文は、解像度を未定義にして同じドキュメントを作成します。

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

注意：プレースホルダー `undefined` は、定義したいパラメーターに「達する」ためだけに使用します。次の文はドキュメントの高さと幅のみを定義しています。後に続くオプションのパラメーターにプレースホルダーは不要です。

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

プレースホルダー `undefined` は大文字と小文字を区別します。

Tell 文（AS のみ）

AppleScript のサンプルは次の文で開始 / 終了します。

```
tell application "アプリケーション名"
end tell
```

`tell` 文は、文内に含まれる全コマンドを実行するデフォルトのオブジェクト名を指定します。前の例で、`tell` 文はアプリケーションオブジェクトを対象とします。したがって、スクリプト文の `tell` 文内で別のオブジェクトが明示的に指定されていなければ、文内に含まれているコマンドすべてをアプリケーションオブジェクトが実行する必要があります。

次のスクリプトは、コマンドがどのオブジェクトを処理するかを示すために、各オブジェクト全体の包含階層を注意深く概説しています。

```
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
```

コマンドの対象を変更してショートカットを作成できます。それには、ネスト構造の `tell` 文を追加します。次のスクリプトは、前述のスクリプトと同じ処理を実行します。ネスト構造の `tell` 文はドキュメントオブジェクトを対象とするため、レイヤーを作成する文でドキュメントオブジェクトを参照する必要がありません。

```
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
```

tell 文は、それぞれが end tell 文で終了する必要があります。

tell 文は、いくつでもネストできます。

変数に関する注意

このセクションでは、変数の使用について情報を補足します。

変数の値の変更

変数の値はいつでも変更できます。変数を変更するには、変数名の後に代入演算子（AS では to、JS と VBS では =）と新しい値を入力します。次のスクリプトは、新しいレイヤーを含む変数 layerRef を作成した後、すぐに第2のレイヤーを作成し、それを layerRef の新しい値として代入します。

AS AS で変数の値を変更するには、set コマンドを使用します。

```
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 JS で変数の値を変更するには、変数名の後に等号（=）と新しい値を入力します。再代入文を var で始めないでください。var は新しい変数を作成する場合にのみ使用します。

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

VBS VBS で変数の値を変更するには、Set コマンドを使用します。

```
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"
```

既存のオブジェクトを参照するための変数の使用

変数を作成して既存のオブジェクトを含めることもできます。

```
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
```

スクリプトファイルを読み易くする方法

このセクションでは、スクリプトファイルを読み易くするオプションを2つ紹介します。

- ▶ コメント
- ▶ 改行

スクリプトのコメント記述

スクリプトのコメントは、スクリプトエンジンがスクリプトの実行時に無視するテキストです。

コメントは、スクリプトの処理や目的を（自分や他者のために）記録するときに役立ちます。上級者を含め、大半のプログラマーはスクリプトのほぼすべての要素にコメントを挿入しています。コメントは、スクリプトを記述している間はさほど重要に見えないかもしれませんが、1か月後または1年後にスクリプトを開いて、どういう理由で何をしようとしていたのかを知りたいとき、コメントを入れておいて良かったと思うはずです。

AS ASで1行の全体または一部にコメントを記述するには、コメントの先頭にハイフンを2つ（--）入力します。複数行にコメントを記述するには、コメントを（*と*）で囲みます。

```
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 don't the rest of your script is
    hidden from the scripting engine!*)
end tell
```

注意：このスクリプトが実行することは、現在のドキュメントの印刷だけです。

JS JSで1行の全体または一部にコメントを記述するには、コメントの先頭にスラッシュを2つ（//）入力します。複数行にコメントを記述するには、コメントを/*と*/で囲みます。

```
//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...*/
```

注意：このスクリプトが実行することは、アクティブなドキュメントの印刷だけです。

VBS VBSでは、コメントの先頭にRem（「remark」の略）または'（一重引用符）を入力します。VBSは、複数行にわたるコメントをサポートしていません。複数行のコメントを連続的に記述するには、各行をどちらかのコメント書式で開始します。

```
'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?
```

注意：このスクリプトが実行することは、アクティブなドキュメントの印刷だけです。

AppleScript と VBScript での長い行の継続

AppleScript と VBScript では、行末のキャリッジリターンで文の終わりを示します。スクリプトの行が長く1行に収まらない場合は、改行してもそれを正しい命令として読み取らせる特殊な継続文字を使用します。

注意：スクリプトエディターのウィンドウを拡大して、文を1行で続けることもできます。

AS 改行を入れて文を続けるには、文字␣（**Option+Return キー**）を入力します。

```
tell application "Adobe InDesign CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc with properties {name:"My First Layer"} at the"␣
beginning of myDoc (* without the line break character, AS would consider this
    line an incomplete statement*)
    (* note that line continuation characters are not required in a multi-line comment
such as this one*)
    set myLayer2 to make layer in myDoc with properties {name:"My Other Layer"} "␣
before myLayer
end tell
```

VBS アンダースコア (_) の後にキャリッジリターンを入力し、長い行を折り返して文を続けます。

注意：この2言語では、継続文字を文字列内（引用符の内側）に置くと、その機能が失われます。文字列内で改行が発生する場合は、改行文字を文字列の前に置いて改行を早めに挿入します。

注意：JavaScript では、文にキャリッジリターンを含めることができますので、継続文字は不要です。ただし、ExtendScript インタプリタでは各行が完全な文と解釈されます。したがって、通常は改行を文の終わりに挿入することをお勧めします。

配列の使用

VBScript と JavaScript では、配列はコレクションと似ていますが、配列は自動的に作成されないという点が異なります。

配列は1つの変数の値のリストと見なすことができます。例えば、次の JavaScript の配列は変数 `myFiles` の4つの値をリストにします。

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

それぞれの値に番号が付いています。文で値を使用するには、この番号を含める必要があります。次の文はファイル `clouds.gif` を開きます。

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

次の例は VBScript における同じ文です。

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

注意：VBS のコレクションのインデックス番号は常に (1) から開始しますが、作成する配列を (1) で始めるか (0) で始めるかを VBS スクリプトで指定できます。配列のインデックス開始番号を設定する方法については、VBScript のマニュアルを参照してください。コレクションとインデックス番号については、[13 ページの「オブジェクト参照としてのオブジェクトコレクションまたはオブジェクト要素」](#)を参照してください。

オブジェクトの作成

最初のスクリプトでは、コレクションオブジェクトの `make` コマンド (AS)、`add()` メソッド (JS)、または `Add` メソッド (VBS) を使用してオブジェクトを作成する方法を紹介します。次に例を示します。

AS

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

JS

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

VBS

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

しかしながら、make コマンド (AS)、add() メソッド (JS)、または Add メソッド (VBS) がないオブジェクトもあります。このタイプのオブジェクトを作成するには、ご使用のアプリケーションの Adobe スクリプティングガイドで、ご使用のスクリプト言語の章にある「オブジェクトの新規作成」の節を参照してください。

スクリプトに関する追加情報

以上で、基本タスクを実行する簡単なスクリプトを作成できるだけの知識が得られたはずです。さらに高度なスクリプト技術を身に付けるには、次の資料を利用してください。

- ▶ [44 ページの「高度なスクリプティング技法」](#)
- ▶ 作成するアプリケーションの Adobe スクリプティングガイド
- ▶ [第6章「参考資料」](#)

3 オブジェクトのプロパティおよびメソッドの検索

アドビ システムズ社は、有効なスクリプトを作成するのに必要なオブジェクト、メソッドやコマンド、プロパティ、列挙、およびパラメーターを検索して使用するのに役立つ以下のリソースを用意しています。

- ▶ オブジェクトディクショナリまたはタイプライブラリ。スクリプト可能な各 Adobe アプリケーションには、スクリプトエディター環境内で使用できる参照ライブラリまたはディクショナリが用意されています。
- ▶ Adobe スクリプティングリファレンス (PDF 形式)。これらのリファレンスはインストール CD に収録されています (スクリプティングリファレンスは、すべての Adobe アプリケーションに用意されているわけではありません)。

スクリプティング環境ブラウザーの使用

この節では、各スクリプティング言語についてスクリプティング環境オブジェクトブラウザーを表示して使用方法を説明します。

AppleScript データディクショナリ

AppleScript ディクショナリは、Apple のスクリプトエディターアプリケーションによって使用できます。

AppleScript ディクショナリの表示

注意: スクリプトエディターアプリケーションはデフォルトでは、アプリケーション /AppleScript/ スクリプトエディターにあります。



1. スクリプトエディターで、ファイル／用語説明を開くを選択します。スクリプトエディターが用語説明を開くダイアログボックスを表示します。
2. Adobe アプリケーションを選択して、「開く」をクリックします。スクリプトエディターは Adobe アプリケーションを開き、そのアプリケーションのディクショナリを表示します。

AppleScript (AS) ディクショナリの使用

AS ディクショナリはオブジェクトをグループに分けます。グループの名前はそのグループに含まれるオブジェクトのタイプを示すものになっています。

オブジェクトのプロパティを表示するには:

1. データディクショナリ画面の左上のパネルで、オブジェクトを含むグループを選択します。
2. 上部中央のパネルでオブジェクトを選択します。

注意: オブジェクトは四角形のアイコン  で示され、コマンドは丸いアイコン  で示されます。

オブジェクトの説明が下部のパネルに表示されます。その説明の下に、オブジェクトの要素とプロパティのリストが表示されます。各要素名は、その要素のオブジェクトタイプに対するハイパーリンクです。

3. プロパティのリストには次のものが含まれます。

- ▷ プロパティ名
- ▷ データ型（丸かっこの中）

データ型がオブジェクトである場合、データ型はそのオブジェクトへのハイパーリンクになります。

データ型が列挙の場合、データ型は「anything」になります。プロパティの説明の後に、「Can return:」に続いて有効な値が示されます。各値はスラッシュ (/) で区切られます。

- ▷ アクセス値：

オブジェクトが読み取り専用プロパティの場合、データ型の後ろに *r/o* と示されます。

オブジェクトが読み書きプロパティである場合、アクセス値は示されません。

- ▷ プロパティの説明

1. グループを選択すると、そのグループのオブジェクトとコマンドが上部中央のパネルに表示されます

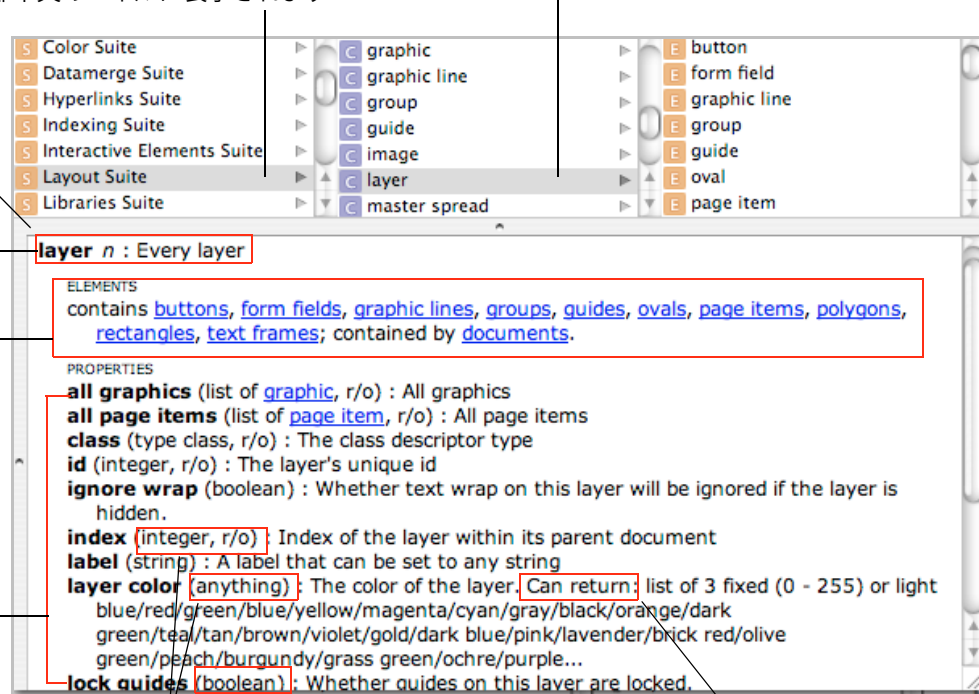
2. オブジェクトを選択します

3. オブジェクトの情報が下部のパネルに表示されます

オブジェクトの説明

オブジェクトの要素へのリンク

プロパティリスト



プロパティ名に続いて、データ型およびアクセス値が丸かっこ付きで示されます。**注意：**アクセス値は、プロパティが読み取り専用の場合にのみ表示されます。

列挙値の前には「Can return:」が付けられます

コマンドおよびコマンドパラメーターの表示



注意：データディクショナリにはコマンドと共に使用できるオブジェクトのリストが表示されます。しかし、オブジェクトと共に使用できるコマンドのリストは表示されません。オブジェクトと共に使用できるコマンドのリストを表示するには、ご使用のアプリケーションの [AppleScript スクリプティングリファレンス](#) を参照してください。詳しくは、[40 ページの「Adobe スクリプティングリファレンスの使用」](#) を参照してください。

データディクショナリでコマンドを表示するには、次の手順に従ってください。

1. データディクショナリ画面の左上のパネルで、コマンドの入ったグループを選択します。

そのグループに入っているコマンドとオブジェクトが上部中央のパネルに表示されます。

2. 上部中央のパネルでコマンドを選択します。

注意：コマンドは丸いアイコン  で示され、オブジェクトは四角形のアイコン  で示されます。

コマンドの説明が下部のパネルに表示されます。

▷ その説明の下に、コマンドと共に使用できるオブジェクトのリストが表示されます。

▷ 使用できるオブジェクトの下にパラメーターのリストが表示されます。

オプションパラメーターは角かっこ（[]）付きで示されます。

パラメーター名に角かっこが付けられていないものは、必須パラメーターです。

▷ 各パラメーター名の後ろにデータ型が示されます。

データ型がオブジェクトである場合、データ型はそのオブジェクトへのハイパーリンクになります。

データ型が列挙である場合、「Can accept:」に続いて有効な値が示されます。各値はスラッシュ（/）で区切られます。

1. グループを選択すると、そのグループのコマンドとオブジェクトが上部中央のパネルに表示されます

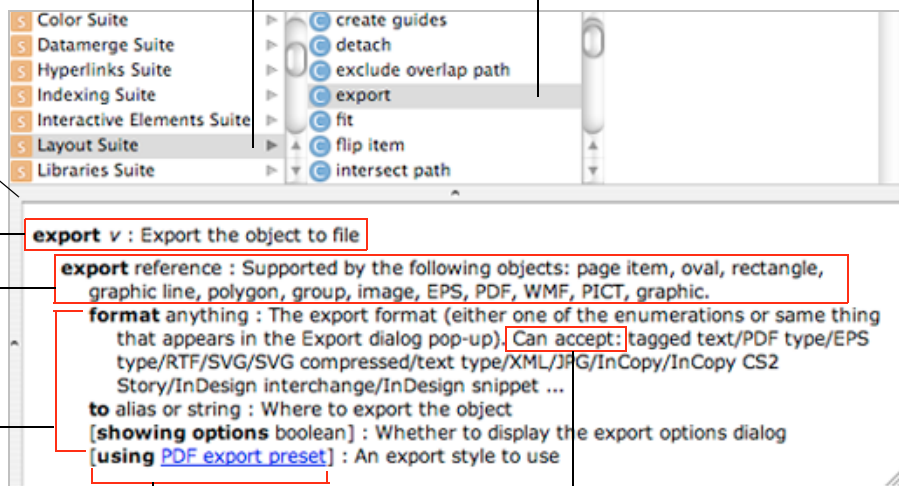
2. コマンドを選択します

3. そのコマンドの情報が下部のパネルに表示されます

コマンドの説明

コマンドを使用するオブジェクトのリスト

パラメーター、データ型、説明



オプションパラメーターは角かっこ（[]）付きで示されます

注意：パラメーター値が列挙の場合、「Can accept:」に続いて列挙値が示されます

JavaScript オブジェクトモデルビューア

Adobe アプリケーションと共にインストールされる ExtendScript Tools Kit (ESTK) を使用して、Adobe アプリケーションで使用できる JavaScript オブジェクトおよびメソッドを表示できます。

Adobe アプリケーションの JavaScript オブジェクトモデルビューアを表示して使用方法については、『JavaScript ツールガイド』を参照してください。

VBScript タイプライブラリ

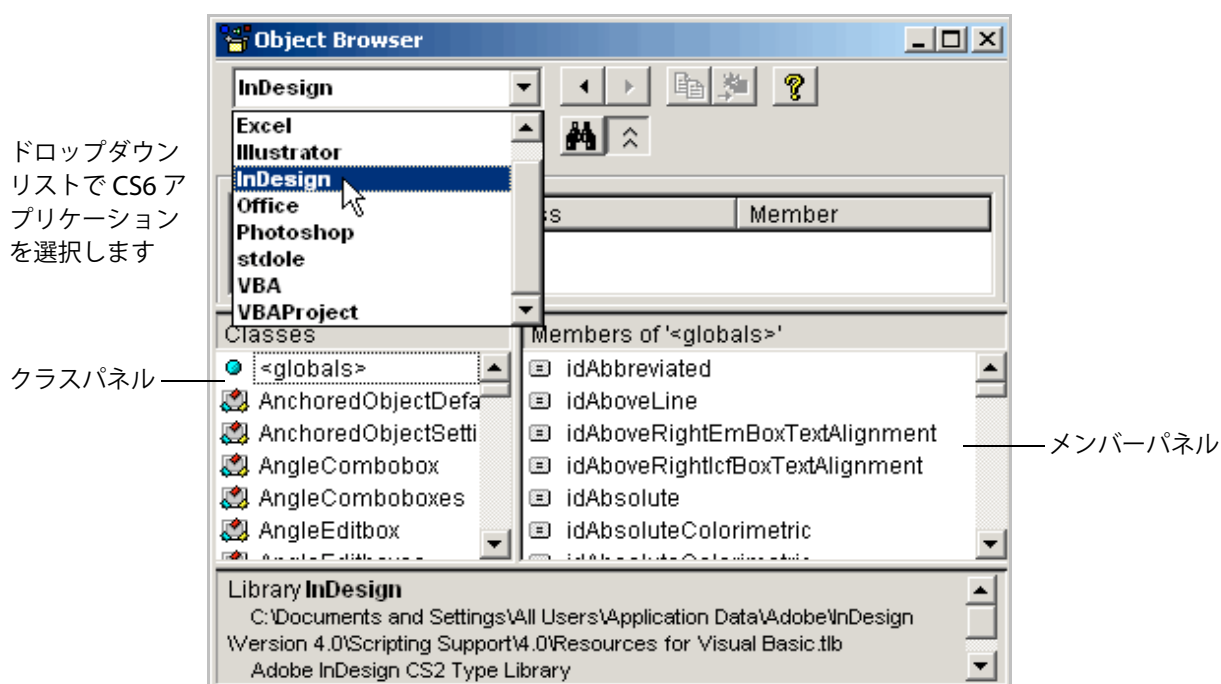
Microsoft Office アプリケーションの Visual Basic エディターを使用して、Adobe アプリケーションで使用できる VBScript オブジェクトおよびメソッドを表示できます。

注意：異なるエディターを使用する場合は、エディターのヘルプシステムを参照してタイプライブラリの表示方法を調べてください。

VBScript タイプライブラリの表示

VBS オブジェクトライブラリを表示するには、次の手順に従ってください。


1. 任意の Microsoft Office アプリケーションで、ツール／マクロ／ Visual Basic Editor を選択します。
2. Visual Basic Editor で、ツール／参照設定を選択します。
3. 参照設定ダイアログの「参照可能なライブラリファイル」で Creative Suite アプリケーションを選択し、「OK」をクリックします。
4. Visual Basic のエディターウィンドウで、表示／オブジェクトブラウザーを選択します。
5. オブジェクトブラウザーウィンドウの左上にあるドロップダウンリストで Adobe アプリケーションを選択します。




VBSript タイプライブラリの使用


VBS オブジェクトタイプライブラリでは、オブジェクトブラウザーウィンドウの左側にあるクラスパネルにオブジェクトと定数が表示されます。クラスパネルでは、次のように表示されます。

▶ オブジェクトは  アイコンで示されます。

▶ 定数は  アイコンで示されます。

オブジェクトのプロパティとメソッドを表示するには、クラスパネルでオブジェクトタイプを選択します。プロパティとメソッドのリストは、クラスパネルの右側にあるメンバーパネルに表示されます。

▶ プロパティは  アイコンで示されます。

▶ メソッドは  アイコンで示されます。

オブジェクトブラウザーのプロパティリストの概要

メンバーパネルでプロパティを選択すると、そのプロパティの情報がオブジェクトブラウザーウィンドウの下部にある情報パネルに次のように表示されます。

▶ プロパティ名の後ろにデータ型が示されます。

- ▷ データ型が定数である場合、その定数は定数値へのハイパーリンクになります。定数名には Adobe アプリケーションの省略名が接頭辞として付けられます。次に例を示します。

Photoshop CS6 の列挙には、接頭辞として Ps が使用されます。

例：PsColorProfileType、PsBitsPerChannelType

InDesign CS6 の列挙には、接頭辞として id が使用されます。

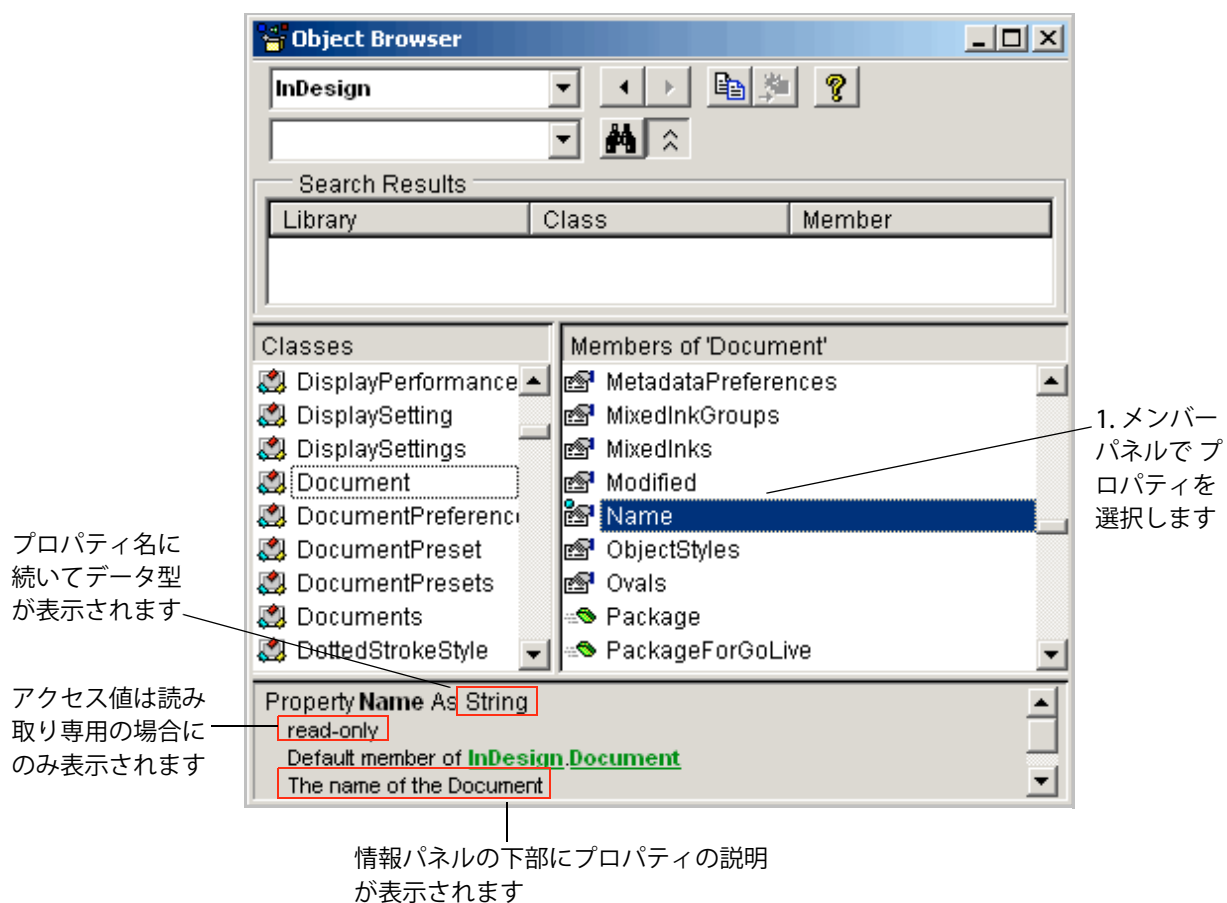
例：idRenderingIntent、idPageOrientation

Adobe Illustrator CS6 の列挙には、接頭辞として Ai が使用されます。

例：AiCropOptions、AiBlendModes

- ▷ データ型がオブジェクトである場合、オブジェクト名はそのオブジェクト型へのハイパーリンクになります。

- ▶ アクセス値は、プロパティが読み取り専用の場合にのみ表示されます。プロパティが読み書き可能な場合、アクセス値は表示されません。



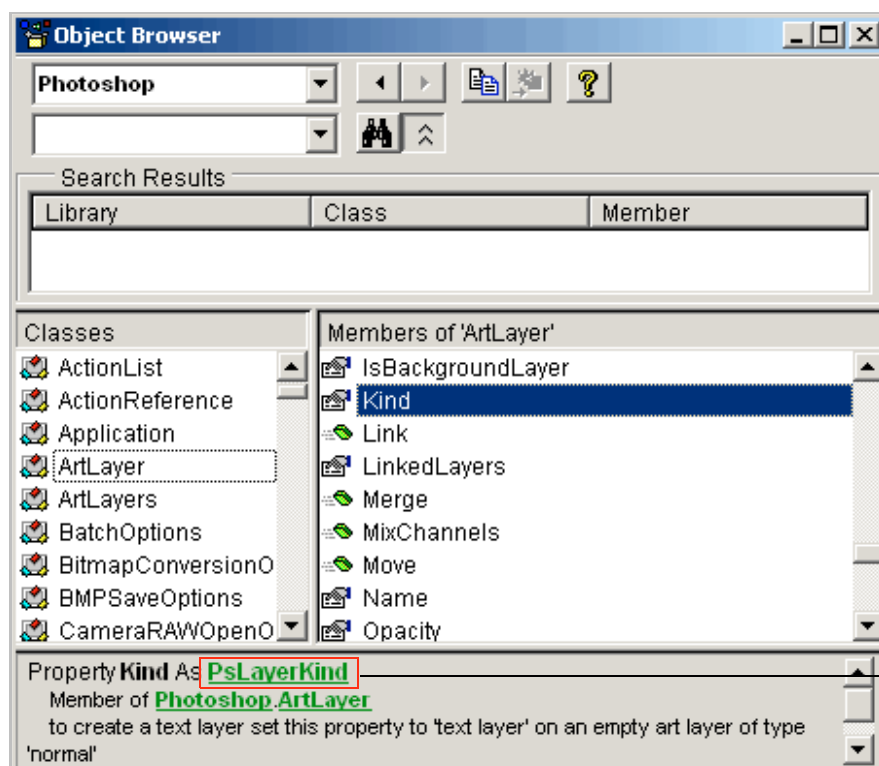
列挙の数値の検索

VBS では、列挙の数値をプロパティ値として使用します。例えば、以下のスクリプトでは、最後の行の Kind プロパティで表されているレイヤータイプは数値 2 として定義されています。これは、TextLayer 定数値を表します。

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

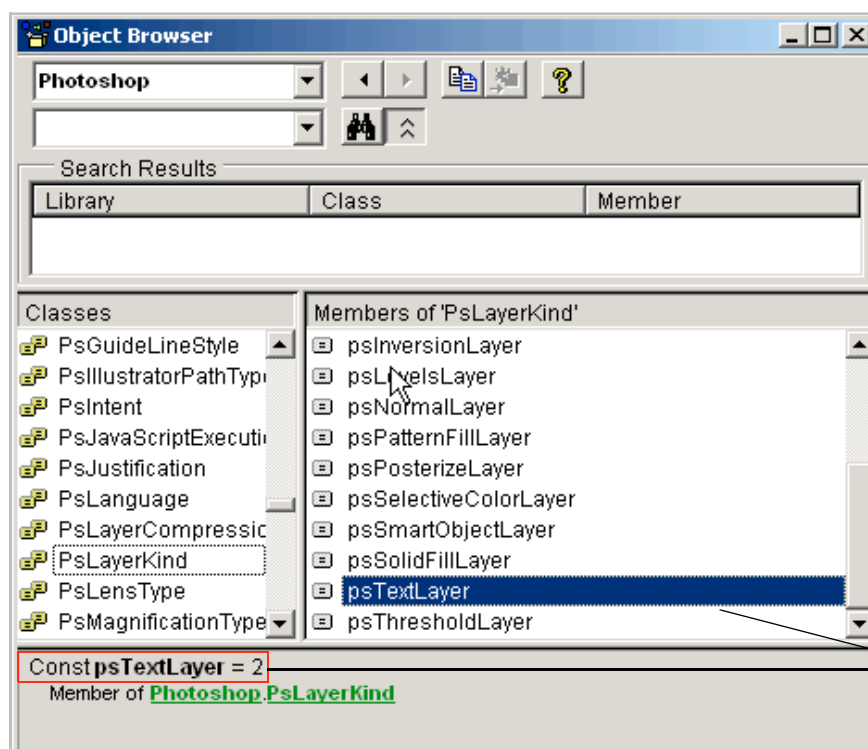
列挙の数値を検索するには、次の手順に従ってください。

1. 列挙の情報に対するリンクをクリックします



列挙の情報に対するリンクをクリックします

2. 列挙値をクリックして、下のパネルに数値を表示します。



右側のパネルで列挙値をクリックして、下のパネルに数値を表示します

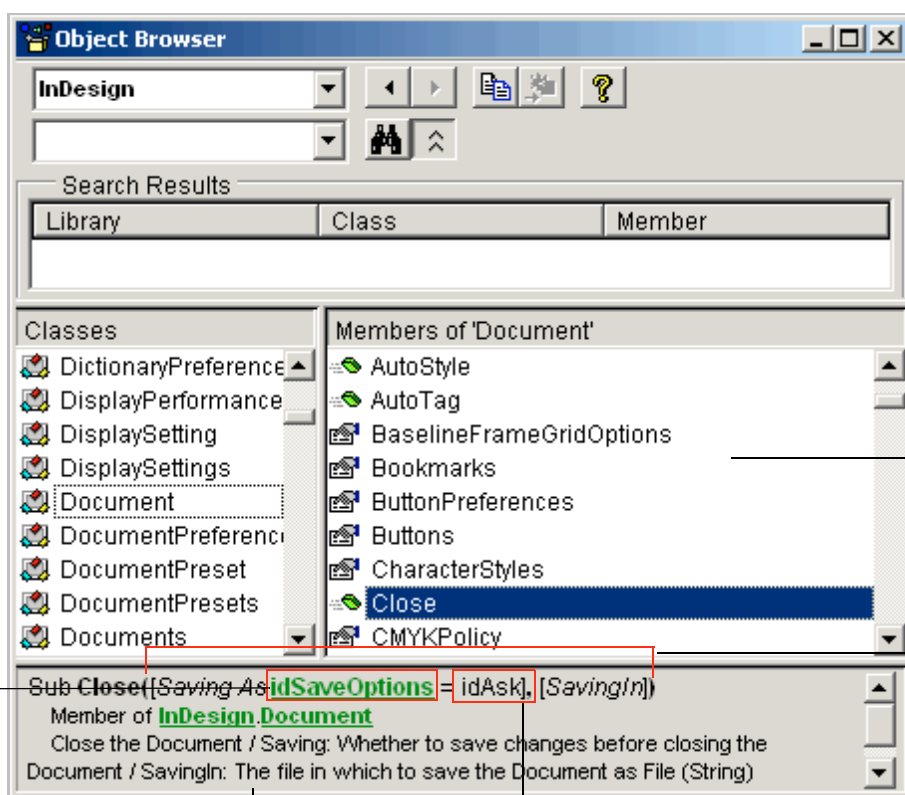
メソッドのリストの概要

メンバーパネルでメソッドを選択すると、そのメソッドの情報がオブジェクトブラウザーウィンドウの下部にある情報パネルに次のように表示されます。

- ▶ メソッド名の後ろにパラメーターが示されます。
 - ▷ オプションパラメーターは角カッコ ([]) 付きで示されます。
 - ▷ パラメーター名に角カッコが付けられていないものは、必須パラメーターです。
- ▶ 各パラメーター名の後ろにデータ型が示されます。
 - ▷ データ型がオブジェクトである場合、データ型はそのオブジェクトへのハイパーリンクになります。
 - ▷ データ型が列挙である場合、列挙名はアプリケーションの頭文字で、列挙の情報に対するハイパーリンクになります。
 - ▷ パラメーターにデフォルト値が用意されている場合、その値がデータ型の後ろの等号 (=) に続いて示されます。

注意：デフォルト値は、パラメーターに値を定義しない場合に使用されます。デフォルト値はオプションパラメーターにのみ用意されています。

メソッド名に続いてデータ型が示されます。データ型が列挙である場合、列挙名はアプリケーションの頭文字で始まり、列挙の情報に対するリンクになります



1. メンバーパネルでメソッドを選択します

メソッド名に続いてパラメーターのリストが角カッコ内に示されます。オプションパラメーターは角カッコ ([]) 付きで示されます

メソッドの説明が情報パネルの下部に表示されます

デフォルト値が用意されている場合、その値が等号 (=) に続いて示されます。**注意：**デフォルト値はすべてのデータ型に指定できます

Adobe スクリプティングリファレンスの使用

アドビ システムズ社では、さまざまなアプリケーションについてスクリプティングリファレンスを用意しています。リファレンスはインストール CD に収録されています。

スクリプティングリファレンスでは、言語ごとに章が分かれています。各章では、オブジェクトはアルファベット順に説明されています。各オブジェクトについて、次に示す表が用意されています。

- ▶ 要素 (AS のみ)
- ▶ プロパティ
- ▶ メソッド、コマンド、または関数

さらに、ほとんどのオブジェクトの節には、オブジェクト、およびそのプロパティとメソッドまたはコマンドを使用したスクリプティングのサンプルが含まれています。サンプルスクリプトは例として使用できます。また、サンプルを基にしてプロパティやメソッドに変更を加えることができます。

オブジェクトの要素表の操作 (AS のみ)

要素は、1 つのオブジェクトに含まれているオブジェクトの集合です。オブジェクトに要素が含まれている場合、表には要素を参照するさまざまな方法が示されます。スクリプティングの初心者の場合、まず要素表の名前列または要素列について理解する必要があります。これらの列は包含階層でオブジェクトの直下にあります。例として、InDesign の document オブジェクトの要素表を以下に示します。

名前	参照先
character style	index、name、range、relative、satisfying a test、ID
layer	index、name、range、relative、satisfying a test、ID
story	index、name、range、relative、satisfying a test、ID

この表からわかることは、このアプリケーションのために作成する document オブジェクトで作成できるのは、character style、layer、および story オブジェクトであるということです。

次に例を示します。

```
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
```

次に示すスクリプト文では、stroke style がアプリケーションの document オブジェクトの要素ではないため、エラーが発生します。

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

オブジェクトのプロパティ表の操作

オブジェクトのプロパティ表には以下が示されます。

▶ オブジェクトと使用できるプロパティ

▶ 各プロパティの値タイプ

値タイプが定数または列挙の場合、値は有効値のリストまたは定数リストへのハイパーリンクとして示されます。

値タイプが別のオブジェクトである場合、その値はオブジェクトのリストへのハイパーリンクとして示されます。

▶ プロパティの入力状況：読み取り専用または読み書き

▶ 説明。次のものを含みます。

▷ プロパティの定義内容の説明

▷ 有効値の範囲

▷ 他のプロパティとの依存関係

Photoshop の art layer オブジェクトのサンプルプロパティ表を以下に示します。この表には、データの各タイプのサンプルが掲載されています。

プロパティ	値タイプ	説明
bounds	4 つの数字の配列	読み取り専用。レイヤーの矩形の境界を記述する [y1, x1, y2, x2] 形式の座標の配列。
kind	LayerKind	読み取り専用。レイヤーのタイプ。
name	string	読み書き。レイヤーの名前。
opacity	number (double)	読み書き。不透明度のパーセント（範囲：0.0 ～ 100.0）。
textItem	TextItem オブジェクト	読み取り専用。レイヤーに関連付けられているテキスト項目。 注意： kind = LayerKind.TEXT の場合のみ有効。kind を参照してください。
visible	Boolean	読み書き。真（true）の場合、レイヤーは表示されます。

次に例を示します。

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
```

注意：bounds プロパティが読み取り専用であるため、レイヤーの境界を定義できません。

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
```

注意：JS および VBS では、集合オブジェクトはそれを含むオブジェクトのプロパティで管理されます。オブジェクトの包含階層を決定するには、オブジェクトまたはそのオブジェクトの集合オブジェクトを使用するオブジェクト（オブジェクトの複数形）をプロパティとして配置する必要があります。例えば、documents.layers または layers.textFrames のようにします。

オブジェクトのメソッド表の操作

オブジェクトのメソッド表には以下が示されます。

- ▶ オブジェクトと使用できるメソッド
- ▶ 各メソッドのパラメーター（複数も可）
 - ▷ パラメータータイプが定数または別のオブジェクトである場合、その値は定数またはオブジェクトのリストへのハイパーリンクとして示されます。以下に示すサンプルのメソッド表では、パラメータータイプ NewDocumentMode と DocumentFill は定数です。
 - ▷ パラメーターには必須とオプションがあります。オプションパラメーターは角かっこ（[]）付きで示されます。
- ▶ メソッドが生成する戻り値のタイプ。

定数またはオブジェクトが戻される場合、その値は定数またはオブジェクトのリストへのハイパーリンクとして示されます。以下に示すサンプルのメソッド表では、戻り値 Document はオブジェクトです。
- ▶ メソッドの実行内容を定義する説明。

以下に示すサンプルのメソッド表は、Photoshop CS6 ドキュメントの add メソッドのパラメーターを示しています。

メソッド	パラメータータイプ	戻り	実行内容
add		Document	ドキュメントオブジェクトを追加します。
[width]	UnitValue		
[, height]	UnitValue		
[, resolution])	number (double)		(pixelAspectRatio
[, name]	string		範囲：0.10 ～ 10.00)
[, mode])	NewDocumentMode		
[, initialFill]	DocumentFill		
[, pixelAspectRatio])	number (double)		

上記の表の説明：

- ▶ パラメーターはすべてオプションパラメーターであり、角かっこ付きで示されています。
- ▶ デフォルトでは、width および height パラメーターは現在の定規単位に解釈されるため、データ型は UnitValue として示されます。つまり、現在の縦方向の定規の単位がインチであり、横方向の定規の単位がセンチメートルである場合、次に示す文は幅が 5 センチメートルで高さ 7 インチのドキュメントを作成します。

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

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

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

- ▶ mode と initialFill は定数値を取得します。

以下に示すスクリプト文は、サンプルのメソッド表にリストされている各パラメーターの値を定義します。

AS make document 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 高度なスクリプティング技法

ほとんどのスクリプトは、最初から最後まで順次に進行するわけではありません。多くの場合、スクリプトは現在のドキュメントから集めたデータによってパスを変えたり、コマンドを複数回繰り返したりします。制御構造は、こうしたことをスクリプトで可能にするスクリプト言語機能です。

条件文

if 文

皆さんが Adobe アプリケーションと話せたら、「ドキュメントに含まれるレイヤーが 1 つのみの場合は別のレイヤーを作成する」と指示する状況があるかもしれません。例えばこれが条件文です。条件文は判断をするためのものです。条件文を使用すると、何か（例えばレイヤーの数）を判断してその結果に従ってアクションを実行するスクリプトを記述できます。条件が一致すると、スクリプトは if 文に含まれるアクションを実行します。条件が一致しなかった場合、スクリプトは if 文に含まれるアクションをスキップします。

以下に示すスクリプトはいずれも、ドキュメントを開いて、ドキュメントに含まれるレイヤーが 1 つだけかどうかをチェックします。存在するレイヤーが 1 つだけだった場合、スクリプトはレイヤーを追加し、新しいレイヤーの塗りつぶし不透明度を 65% に設定します。

AS AS の if 文は if で始まり、その後に丸かっこで囲まれた比較句、次に then が続きます。if 文は end if で閉じる必要があります。

```
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
```

注意: AS では、値の比較に単一等号 (=) を使用します。

ここで Ducky.tif を閉じてもう一度このスクリプトを試してみましょう。ただし、if 文を次のように変更します。

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

JS JS の if 文は if で始まり、その後に丸かっこで囲まれた比較句が続きます。if 文内のアクションを波かっこ ({}) で囲みます。

```
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
}
```

注意：JavaScript では、値の比較に二重等号 (==) を使用します。単一等号 (=) は、値をプロパティまたは変数に代入するために使用します。

ここで Ducky.tif を閉じてもう一度このスクリプトを試してみましょう。ただし、if 文を次のように変更します。

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

VBS VBS の if 文は If で始まり、その後に比較句、次に Then と続きます。if 文は End If で閉じる必要があります。

```
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
```

注意：VBS では、値の比較と割り当ての両方に単一等号を使用します。

ここで Ducky.tif を閉じてもう一度このスクリプトを試してみましょう。ただし、if 文を次のように変更します。

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

if else 文

もう少し複雑な要求があることもあります。例えば、「ドキュメントに含まれるレイヤーが1つの場合はレイヤーの塗りつぶし不透明度を50%に設定し、ドキュメントに含まれるレイヤーが2つ以上の場合はアクティブレイヤーの塗りつぶし不透明度を65%に設定する」という要求です。この種の状況では、if else 文を必要とします。

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
```

ループ

スクリプトで特定のタイプの全オブジェクトを検出して変更することができます。例えば、ドキュメントに可視レイヤーと不可視レイヤーがある可能性があり、すべてのレイヤーを可視にする場合です。このスクリプトを複数のドキュメントに対して機能させたくても、ドキュメントによってレイヤーの数が異なります。

この状況では、repeat 文 (AS) またはループ (JS および VBS) が便利です。ループはオブジェクトのコレクションを「ウォーク」スルーし、各オブジェクトに対してアクションを実行します。

この節で示すスクリプトを使用するには、Adobe アプリケーションを開き、レイヤーが少なくとも 9 つあるドキュメントを作成します。いくつかのレイヤーを可視にし、残りのレイヤーを不可視にします。ドキュメントを保存し、スクリプトを実行します。このとき、アプリケーションの名前と、アプリケーションの DOM 内の layer オブジェクト名を置き換えます。

これらの各ループの背景にある基本原理は、次のとおりです。スクリプトが要素またはコレクション内の最初のレイヤーを識別してレイヤーの可視性を true に設定します。その後、次のレイヤーを識別してこのアクションを繰り返します。各レイヤーに対してアクションが実行されるまで、スクリプトは次のレイヤーを識別します。

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
```

このスクリプトでは、myLayerCount と myCounter という 2 つの変数を使用して、レイヤーを識別した後にレイヤー番号を 1 つずつ増やしていくことをドキュメント内のすべてのレイヤーが識別されるまで繰り返します。

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}
```

このスクリプトでは for ループを使用します。このループは、JavaScript の最も一般的な技法の 1 つです。先の AppleScript と同様に、このスクリプトでは myLayerCount と myCounter という 2 つの変数を使用して、レイヤーを識別した後にレイヤー番号を 1 つずつ増やしていくことをドキュメント内のすべてのレイヤーが識別されるまで繰り返します。増分は、for 文内の 3 番目の文 myCounter++ で行われます。++ 構文は現在値に 1 を加えますが、1 を加えるのはループのアクションが完了してからです。

このスクリプトの for ループをわかりやすく説明すると、以下のようになります。

1. myCounter の値を 0 から始めます。
2. myCounter の値が myLayerCount の値より小さければ、myLayer に割り当てるレイヤーのインデックスとして myCounter の値を使用し、myLayer の可視性を true に設定します。
3. myCounter の値に 1 を加え、myCounter の新しい値を myLayerCount の値と比較します。
4. myCounter がまだ myLayerCount より小さければ、myCounter の新しい値を myLayer のインデックスとして使用し、myLayer の可視性を true に設定した後 myCounter の値に 1 を加えます。
5. myCounter が myLayerCount より小さくなるまで繰り返します。

VBS

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

VBScript の For Each Next ループでは、アクティブドキュメントの Layers コレクション内の各オブジェクトの Visible プロパティを True に設定するようアプリケーションに指示するだけです。コレクションは、親オブジェクトの包含階層（この場合は変数 myDoc）とそれに続くオブジェクト名の複数形であるコレクション名（この場合は Layers）で識別されます。

注意：ループ内で指定するオブジェクトは何でもかまいません。このスクリプトは、次のスクリプトのように object を x に置き換えても同じように機能します。

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

For Each x in myDoc.Layers

    x.Visible = True

Next
```

スクリプトに関する追加情報

各スクリプティング言語には、強力な複雑なスクリプトを記述できる仕掛けと技法がさらに多数含まれています。引き続き Adobe アプリケーションのスクリプティング方法を習得するには、お使いのアプリケーションの Adobe スクリプティングガイドを参照してください。また、[第6章「参考資料」](#)を参照してください。

5 トラブルシューティング

この章では、スクリプト実行時に表示される可能性のあるいくつかの基本的なエラーメッセージの解釈について説明します。

予約語

スクリプトエディターと ESTK では、他の多くのスクリプトエディターと同様に、特定の語を入力するとそれが強調表示されます。

例えば、ブール値の `true` と `false` は、常に強調表示されます。その他の例としては、次のようなものがあります。

AS	<code>tell</code> <code>end</code> <code>with</code> <code>set</code>
JS	<code>var</code> <code>if</code> <code>else</code> <code>with</code>
VBS	<code>Dim</code> <code>Set</code> <code>MsgBox</code>

強調表示されるこれらの語は、特殊な目的のためにスクリプティング言語によって予約されているもので、変数名として使用することはできません。予約語を文字列の一部として使用することはできます。この場合は予約語が引用符で囲まれるからです。コメントの中で使用することもできます。スクリプティングエンジンはコメントを無視するからです。

スクリプトが構文エラーになった場合は、予約語を不適切に使用していないか確認してください。スクリプティング言語の予約語に関する詳細リストは、[第 6 章「参考資料」](#)に記載されたいずれかの情報リソースを参照してください。

AppleScript スクリプトエディターのエラーメッセージ

AppleScript スクリプトにエラーが含まれている場合、スクリプトエディターはスクリプトの問題のある部分を強調表示してエラーメッセージを表示します。

スクリプトの強調表示された部分のスペルと句読点を確認してください。強調表示されたテキストにエラーが見つからない場合は、強調表示部分の直前のテキストを確認します。直前のテキストにエラーが含まれている場合、それが原因で、スクリプトエンジンは検出部分と異なるテキストを強調表示する可能性があります。

一般的なエラーメッセージのいくつかを、以下に説明します。

- ▶ **Can't get object** : 多くの場合、オブジェクトが包含階層で適切に定義されていません。スクリプト内のオブジェクト名の後に *parent-object* (*parent-object* はエラーメッセージで示されたオブジェクトを含むオブジェクト) を追加してみるか、親オブジェクトを指定するネストされた `tell` 文を作成してください。
- ▶ **Expected "" but found end of script** : すべての引用符が文字列を囲んで閉じていることを確認してください。
- ▶ **Requested property not available for this object** : すべてのプロパティのスペルを確認してください。

ヒント : スクリプトエディターウィンドウの下部にある「イベントログ」を選択すると、スクリプトの進行を1行ずつ調べることができます。

ESTK のエラーメッセージ

ESTK では、以下のようにいくつかの方法でエラーが警告されます。

- ▶ スクリプトに構文エラーが含まれている場合、スクリプトは実行されず、スクリプトの問題のある部分がグレーで強調表示されます。多くの場合、問題の説明が ESTK ウィンドウの下部のステータスバーに表示されます。

構文エラーになった場合は、以下について確認してください。

- ▷ 大文字小文字を正しく使用していることを確認します。JavaScript のすべての用語（列挙名を除く）は小文字で始まり、複合用語の各語の最初の文字には大文字を使用します（例えば `artLayer`）。
- また、変数名は大文字と小文字が区別されます。
- ▷ 丸括弧、波括弧、引用符をすべて閉じます。これらをそれぞれ対にしておく必要があります。
- ▷ 引用符がストレート引用符であることを確認します。また、一重引用符と二重引用符を混用しないようにします。次に例を示します。

誤 : `myDoc.name = "My Document"`

正 : `myDoc.name = 'My Document'`

正 : `myDoc.name = "My Document"`

注意 : スマート引用符など、構文エラーによっては赤で強調表示されるものもあります。ステータスバーのメッセージには「構文エラー」と表示されるだけです。スマート引用符でない引用符を使用するようにしてください。

- ▶ オブジェクトが正しく識別されない、またはオブジェクトが使用するプロパティが存在しないなどのランタイムエラーがスクリプトに含まれている場合、問題のあるスクリプトは強調表示されますが、右下隅に渦巻きアイコンが表示されてスクリプトは実行され続けます。また、エラーは JavaScript コンソールペインとステータスバーの両方に示されます。

ランタイムエラーが発生したら、以下のようにします。

- ▷ **デバッグ／停止**を選択するか **Shift+F5** キーを押してスクリプトを停止します。
- ▷ JavaScript コンソールの表示を確認して、エラーの性質を調べます。以下に、一般的なエラーメッセージのいくつかについて、エラー解決の着手ポイントがわかるように簡潔に説明します。

element is undefined：未定義要素が変数の場合は、変数名のスペルが正しいことと大文字小文字を正しく使用していることを確認します。また、変数が `var` 文を使用して定義されているか、変数に値が代入されていることを確認します。

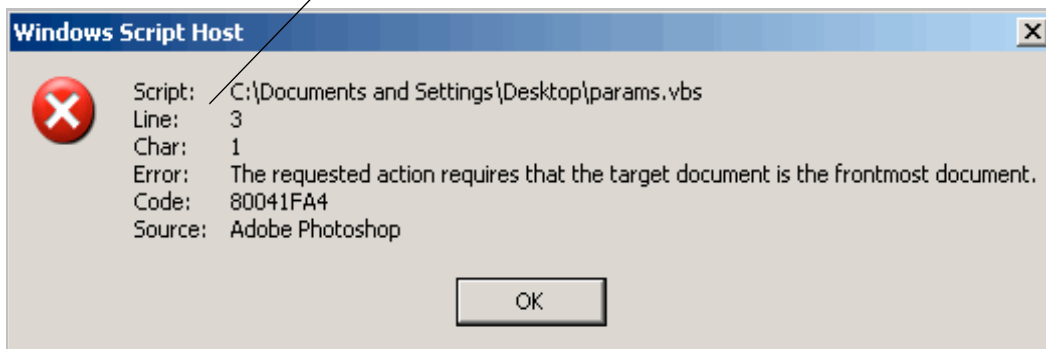
未定義要素が文字列値である場合は、値が引用符で囲まれていることを確認します。

undefined is not an object：強調表示された文内のオブジェクトが包含階層で正しく識別されているかどうかを確認します。例えば、オブジェクトがレイヤーの場合は、そのレイヤーが含まれるドキュメントを定義していることを確認します。ドキュメントオブジェクトの場合は、親オブジェクト `app` を含めることが必要な場合があります。

VBScript のエラーメッセージ

VBScript スクリプトにエラーが含まれている場合、Windows スクリプトホストにエラーメッセージが表示されます。このエラーメッセージでは、エラーが発生した行と、問題のある構文またはオブジェクトが始まる行内の位置が示されます。

このメッセージは、スクリプトの行 3 の最初に問題があることを示しています。



6 参考資料

この章には、スクリプティングの初心者を対象とした資料のリストを記載します。ここに記載されているのは、一部にすぎません。インターネットを検索すれば、ご使用のスクリプト言語のオンラインチュートリアルを見つけることができます。

AppleScript

AppleScript スクリプト言語について詳しくは、以下の文書およびリソースを参照してください。

- ▶ 『AppleScript for the Internet: Visual QuickStart Guide』（初版）Ethan Wilde 著、Peachpit Press 出版、1998 年。ISBN 0-201-35359-8
- ▶ 『AppleScript Language Guide: English Dialect』（初版）Apple Inc. 著、Addison-Wesley Publishing Co. 出版、1993 年。ISBN 0-201-40735-3
- ▶ 『Danny Goodman's AppleScript Handbook』（第 2 版）Danny Goodman 著、iUniverse 出版、1998 年。ISBN 0-966-55141-9
- ▶ Apple Inc. の AppleScript Web サイト：
www.apple.com/applescript

JavaScript

JavaScript スクリプト言語について詳しくは、以下の文書およびリソースを参照してください。

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

VBScript

VBScript および VBA スクリプト言語について詳しくは、以下の文書およびリソースを参照してください。

- ▶ 『Learn to Program with VBScript 6』（初版）John Smiley 著、Active Path 出版、1998 年。ISBN 1-902-74500-0
- ▶ 『Microsoft VBScript 6.0 Professional』（初版）Michael Halvorson 著、Microsoft Press 出版、1998 年。ISBN 1-572-31809-0
- ▶ 『VB & VBA in a Nutshell』（初版）Paul Lomax 著、O'Reilly 出版、1998 年。ISBN 1-56592-358-8
- ▶ Microsoft Developers Network (MSDN) スクリプティング Web サイト：
msdn.microsoft.com/scripting

索引

A

AppleScript
 Web サイト, 51
 最初のスクリプト, 8
 定義, 6
 ディクショナリ, 32

D

DOM
 定義, 10
 表示, 10

E

ESTK
 JS オブジェクトモデルの表示, 35
 デフォルトの場所, 7
 トラブルシューティング, 49
ExtendScript
 定義, 7

I

if else 文, 45
if 文, 44
Illustrator、「Adobe Illustrator」を参照

J

JavaScript
 大文字と小文字の使用法, 15
 最初のスクリプト, 8
 定義, 7
 利点, 7
JavaScript ツールガイド, 7

T

tell 文 (AS), 26

V

var, 11
VBScript
 拡張子, 8
 最初のスクリプト, 8
 タイプライブラリ, 35
 定義, 6

あ

アクション, 5
アラートボックス, 19

い

インデックス
 定義, 13
 番号を付ける方法, 14

お

オブジェクト
 AS ディクショナリでの表示, 32, 34
 VBS タイプライブラリでの表示, 36
 アクティブ, 15
 親, 10
 現在, 15
 コレクション, 13
 参照, 10
 使用, 9
 スクリプティングリファレンスでの表記, 40
 定義, 9
 要素, 13
親オブジェクト, 10

き

起動フォルダー, 7

こ

コマンド
 AS ディクショナリでの表示, 32, 34
 使用, 23
 プロパティ, 23
コメント, 28

さ

参考資料, 51

し

条件文, 44

す

スクリプティング

使用, 5

説明, 6

定義, 6

スクリプト

自動的な実行, 7

スクリプトエディター

AppleScript ディクショナリ, 32

デフォルトの場所, 6

トラブルシューティング, 49

スクリプトのコメント, 28

た

ダイアログ, 19

て

データ型, 16

ディクショナリ, 32

定数

使用, 21

定義済み, 16

な

長い行, 29

は

配列, 30

作成, 30

定義済み, 17

パラメーター

オプション, 24

使用, 24

スクリプティングリファレンスでの表記, 42

ダイレクト (AS), 25

定義, 9

必須, 24

複数の使用, 25

ラベル付き (AS), 25

ひ

引数

使用, 23, 24

定義, 9

ふ

ブール値, 16

プロパティ

AS ディクショナリでの表示, 32

VBS タイプライブラリでの表示, 36

使用, 16

スクリプティングリファレンスでの表記, 41

データ型, 16

定義, 9

複数の値, 17

読み書き, 19

読み取り専用, 19

へ

変数

値の定義, 11

値の変更, 27

既存のオブジェクト, 28

作成, 11

定義, 11

プロパティ値, 16

プロパティ値の使用, 22

命名, 12

ほ

包含階層, 10, 12

スクリプティングリファレンス, 40

ま

マクロ, 5

め

メソッド

VBS タイプライブラリでの表示, 39

使用, 23

スクリプティングリファレンスでの表記, 42

定義, 9

引数, 23

も

文字列, 16

よ

要素

スクリプティングリファレンスでの表記, 40

る

ループ, 46

れ

列挙

使用, 21

定義済み, 16