



Programming Acrobat JavaScript Using Visual Basic

Technical Note #5417


バージョン：Acrobat 6.0



アドビ システムズ株式会社
Adobe Solutions Network

<http://partners.adobe.com/asn/japan/developer/benefits.jsp>

2003 年 5 月



Copyright 2003 Adobe Systems Incorporated. All rights reserved.

注意：All information contained herein is the property of Adobe Systems Incorporated. この出版物（ハードコピー、電子データともに）はすべて、電子的、機械的、複製、録音、その他いかなる形式あるいは手段によっても、事前に発行者の文書による許可を受けることなく、複製または転写することはできません。

PostScript は、アドビ システムズ社の登録商標です。本文中で使用されている PostScript という名称はすべて、特に明記していない限り、アドビ システムズ社により定義された PostScript 言語を指します。PostScript という名称は、アドビ システムズ社が考案した PostScript 言語インタプリタ製品の製品商標としても使用されています。

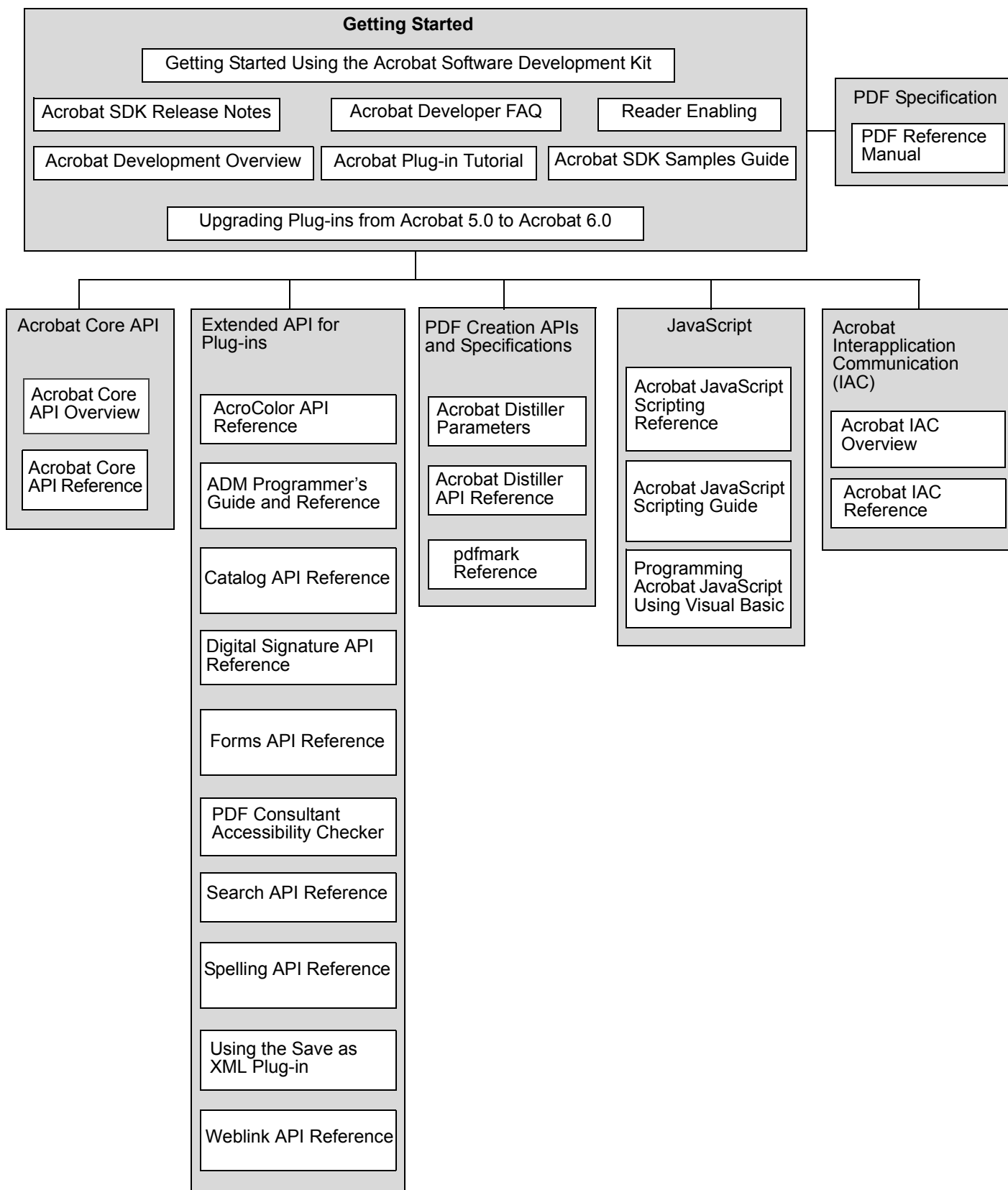
特に指定されている場合を除き、「PostScript プリンタデバイス」、「PostScript ディスプレイデバイス」または同様の記述は、アドビ システムズ社により考案またはライセンス化された PostScript 技術を含むプリンタデバイス、ディスプレイデバイス、またはその他の製品を指し、単に PostScript 言語と互換性があるだけのデバイスあるいはその他の製品を指すものではありません。

Adobe、Adobe ロゴ、Acrobat、Acrobat ロゴ、Acrobat Capture、Distiller、PostScript、PostScript ロゴおよび Reader は、アドビ システムズ社の米国およびその他の国における登録商標または商標です。

Apple、Macintosh および Power Macintosh は、Apple Computer, Inc. の米国およびその他の国における登録商標です。PowerPC は、IBM Corporation の米国における登録商標です。ActiveX、Microsoft、Windows および Windows NT は、米国およびその他の国における Microsoft Corporation の登録商標または商標です。UNIX は、The Open Group の登録商標です。その他すべての商標は、各所有者に所有権があります。

この出版物およびその記載情報については、現時点の状況で提供されており、予告なしに変更される可能性があります。アドビ システムズ社は、この出版物に関して、間違いまたは不整合についての責任および義務を負いません。また、いかなる種類（明示的、非明示的、または法的）の保証も行いません。商業性、特別な用途への適合、および第三者の権利の非侵害に関しても、すべての保証を明示的に否認します。

Acrobat SDK Documentation Roadmap



Programming Acrobat JavaScript Using Visual Basic

Acrobat 6.0 には、Acrobat 環境内で使用できるように設計された JavaScript プログラミングインタフェースの豊富なセットが用意されています。また、外部クライアントが Visual Basic などの環境から同じ機能にアクセスするためのメカニズム (JSObject) も用意されています。

このマニュアルでは、Visual Basic プログラミング環境で JavaScript の拡張機能を使用するために必要な情報を提供します。また、主な概念を説明するための例も示します。

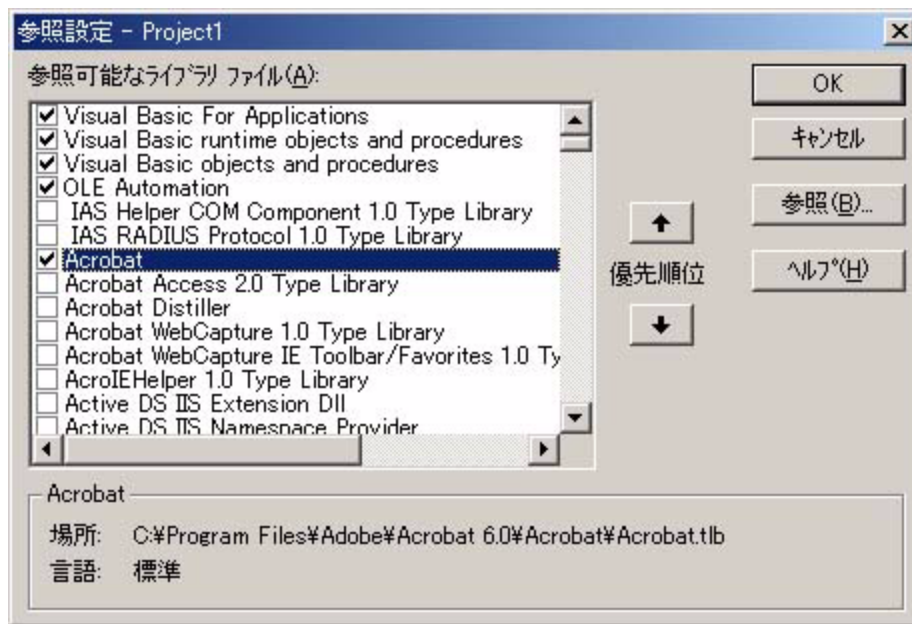
JSObject とは

厳密に言うと、JSObject とは、OLE オートメーションクライアント (Visual Basic アプリケーションなど) と Acrobat JavaScript 機能との間の解釈を行うレイヤーのことです。プログラムの視点から見ると、Visual Basic 環境における JSObject のプログラミングは、Acrobat コンソールを使用した JavaScript でのプログラミングとほとんど変わりありません。

はじめに

次の手順に従って、例を実行するための設定を行ってください。

1. Acrobat 6.0 および Visual Basic 6.0 をインストールします (このマニュアルの例では、両方とも必要になります)。
2. Visual Basic 開発環境を起動します。起動時に**新しいプロジェクト**ダイアログボックスが表示されたら、選択ボックスから「**標準 EXE**」を選択して、「**開く**」をクリックします。空白のフォームとプロジェクトワークスペースが表示されます。
3. JSObject などの Acrobat オートメーション API にアクセスするには、Acrobat のタイプライブラリへの参照を追加する必要があります。メインメニューから**プロジェクト/参照設定 ...**を選択し、使用可能な参照のリストで「Acrobat」というラベルのアイテムにチェックマークを付けます。



簡単な例

この例では、Acrobat の JavaScript コンソールに "Hello, Acrobat!" と表示するために最低限必要なことを説明します。

1. メインメニューから**表示**/**コード**を選択して、このフォームのソースコードウィンドウを表示します。
2. このウィンドウの左上隅にある選択ボックスで「**Form**」を選択します。
右上にある選択ボックスに、Form オブジェクトで使用可能な関数がすべて表示されます。
3. このボックスから「**Load**」を選択すると、中身が空の関数が作成されます。Form が初めて表示されるときには Form の Load 関数が呼び出されるので、この関数は初期化コードを追加するのに適しています。

このプログラムでは、プログラムの動作中に必要となるデータを格納するために、いくつかのグローバル変数を使用します。これらのグローバル変数は **Form_Load** 関数で初期化されます。

例 1 "Hello, Acrobat!"

```
Dim gApp As Acrobat.CAcroApp
Dim gPDDoc As Acrobat.CAcroPDDoc
Dim jso As Object
```

```
Private Sub Form_Load()  
    Set gApp = CreateObject("AcroExch.App")  
    Set gPDDoc = CreateObject("AcroExch.PDDoc")  
    If gPDDoc.Open("c:\adobe.pdf") Then  
        Set jso = gPDDoc.GetJSObject  
        jso.console.Show  
        jso.console.Clear  
        jso.console.println ("Hello, Acrobat!")  
        gApp.Show  
    End If  
End Sub
```

このコードを実行すると、**CreateObject** 呼び出しによって Visual Basic プログラムが Acrobat のオートメーションインタフェイスにアタッチされ、**App** オブジェクトの **Show** コマンドによってメインウィンドウが表示されます。

このコードフラグメントを検討すると、いくつかの疑問が生じるかもしれません。例えば、**gApp** と **gPDDoc** が Acrobat のタイプライブラリに存在する型として宣言されているのに、なぜ **jso** は **Object** として宣言されているのでしょうか。JSObject の固有の型は存在するのでしょうか。

答えは「いいえ」です。**CAcroPDDoc.GetJSObject** 呼び出しのコンテキストを除いて、JSObject はタイプライブラリ内に存在しません。JSObject を通じて JavaScript 機能をエクスポートするとき使用される COM インタフェイスは IDispatch インタフェイスと呼ばれ、Visual Basic では単に「Object」型とされます。つまり、プログラマが使用可能なメソッドは、利用しやすいようにするための明確な定義がされていないこととなります。例えば、次の呼び出しを

```
jso.console.clear
```

次のように置き換えたとして。

```
jso.ThisCantPossiblyCompileCanIt("Yes it can!")
```

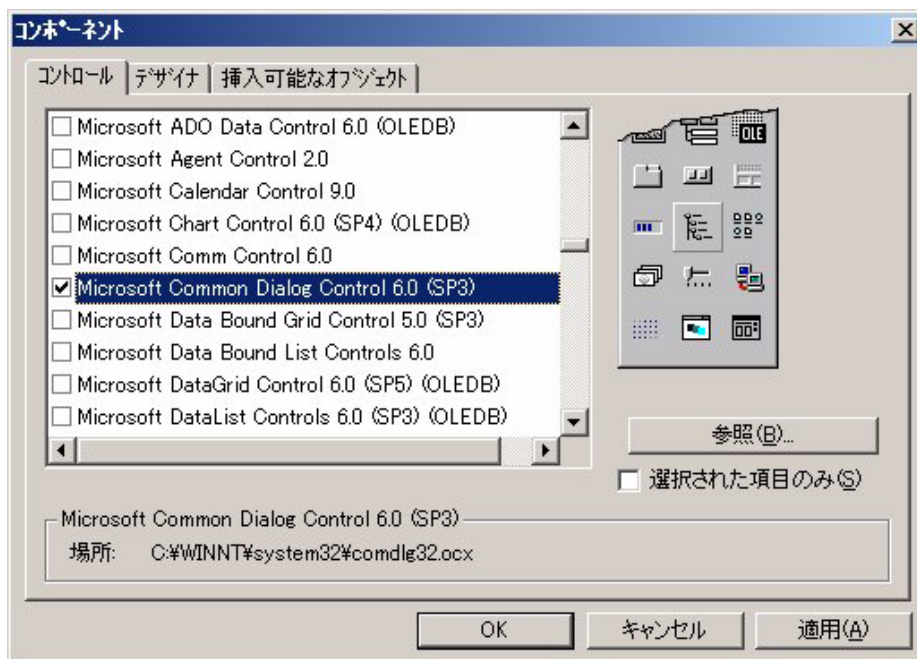
この場合、コンパイラはコードのコンパイルを正常に終わりますが、実行時にエラーを起こします。Visual Basic は JSObject の型情報を持っていないので、特定の呼び出しの構文が有効であるかどうかを実行時まで判断できません。したがって、JSObject の関数呼び出しは無条件にコンパイルされてしまいます。そのため、プログラマはマニュアルを参照して、JSObject インタフェイスを通じて使用できる機能を調べる必要があります。Acrobat SDK に含まれている『[Acrobat JavaScript Scripting Reference](#)』は、JSObject についての理解を深めるために不可欠のマニュアルです。

また、JSObject を作成する前に PDDoc を開く理由についても、疑問を抱くかもしれません。このプログラムを実行しても画面上には文書が表示されませんし、PDDoc を開かずに JavaScript コンソールを表示できるのではないかと思うかもしれません。しかし、使用可能な機能の大半は文書レベルで稼動するため、JSObject は特定の文書と密接に連携して機能するように設計されています。JavaScript (および JSObject) にはアプリケーションレベルの機能もありますが、これらは補助的な機能です。実際には、JSObject は常に特定の文書に関連付けられます。多数の文書进行操作する場合、1 つの JSObject を作成してすべての文書进行操作のではなく、文書ごとに新しい JSObject を取得するようにコードを作成する必要があります。

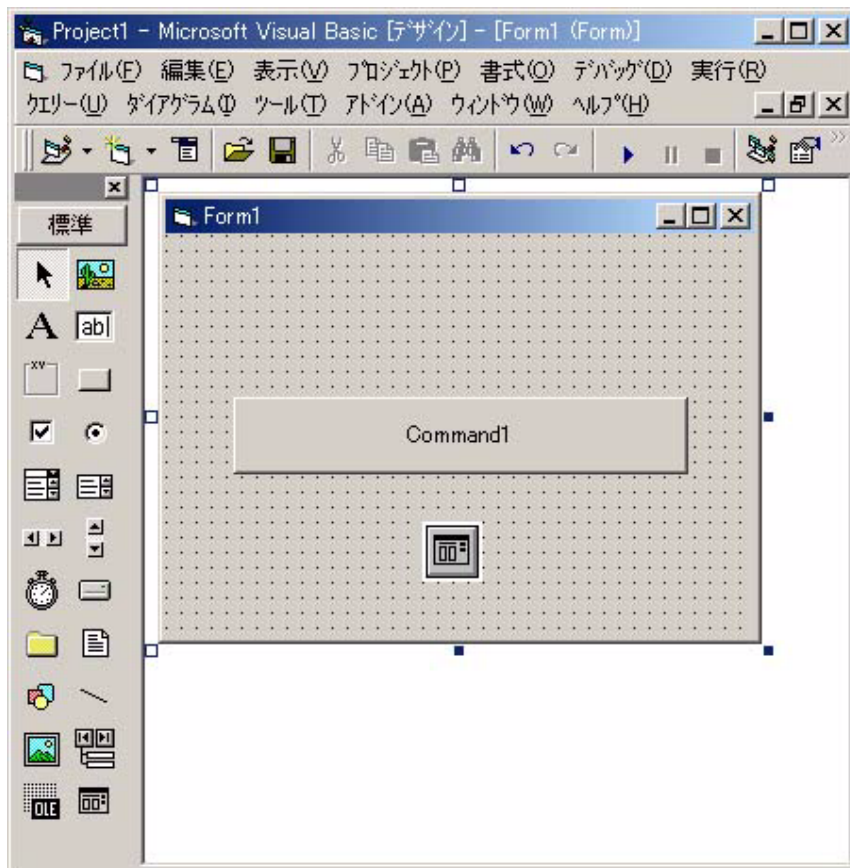
注釈の操作

次に、より興味深い例を示します。このプログラムでは、ユーザが PDF を選択して定義済みの注釈を追加し、ファイルをディスクに保存することができます。

1. メインメニューで**ファイル/新しいプロジェクト**を選択し、「**標準 EXE**」を選択します。
2. この例では、Windows の**ファイル/開く**ダイアログを使用するので、メインメニューから**プロジェクト/コンポーネント…**を選択し、Microsoft Common Dialog Control を追加します。



3. 「**OK**」をクリックし、Common Dialog Control のアイコンがツールバーに追加されていることを確認します。このアイコンを選択してからメインフォーム上でドラッグし、このコントロールをフォームに追加します。同じ方法で、「Command」ボタンをフォームに追加します。



4. これで最小限のユーザインタフェイスの設定が完了したので、メインメニューから表示/コードを選択して、次のソースコードを追加します。

例 2 注釈の操作

```
Dim gApp As Acrobat.CAcroApp

Private Sub Form_Load()
    Set gApp = CreateObject("AcroExch.App")
End Sub

Private Sub Form_Unload(Cancel As Integer)
    If Not gApp Is Nothing Then
        gApp.Exit
    End If
    Set gApp = Nothing
End Sub

Private Sub Command1_Click()
    Dim pdDoc As Acrobat.CAcroPDDoc
    Dim page As Acrobat.CAcroPDPage
    Dim jsO As Object
```

```
Dim path As String
Dim point(0 To 1) As Integer
Dim popupRect(0 To 3) As Integer
Dim pageRect As Object
Dim annot As Object
Dim props As Object

CommonDialog1.ShowOpen
path = CommonDialog1.FileName

Set pdDoc = CreateObject("AcroExch.PDDoc")
If pdDoc.Open(path) Then
    Set jso = pdDoc.GetJSObject
    If Not jso Is Nothing Then

        ' Get size for page 0 and setup arrays
        Set page = pdDoc.AcquirePage(0)
        Set pageRect = page.GetSize
        point(0) = 0
        point(1) = pageRect.y
        popupRect(0) = 0
        popupRect(1) = pageRect.y - 100
        popupRect(2) = 200
        popupRect(3) = pageRect.y

        ' Create a new text annot
        Set annot = jso.AddAnnot
        Set props = annot.getProps
        props.Type = "Text"
        annot.setProps props

        ' Fill in a few fields
        Set props = annot.getProps
        props.page = 0
        props.point = point
        props.popupRect = popupRect
        props.author = "Rob McAfee"
        props.noteIcon = "Comment"
        props.strokeColor = jso.Color.red
        props.Contents = "I added this comment from Visual Basic!"
        annot.setProps props
        pdDoc.Save PDSaveIncremental, path
    End If
    pdDoc.Close
    MsgBox "Annotation added to " & path
Else
    MsgBox "Failed to open " & path
End If

Set pdDoc = Nothing
End Sub
```

この `Form_Load` および `Form_Unload` 関数のコードでは、Acrobat オートメーションインタフェースの初期化とシャットダウンだけを行います。重要な処理はすべて、「Command」ボタンの Click 関数で行われます。最初の数行でローカル変数を宣言し、注釈を付けるファイルを選択するために Windows の開くダイアログを表示します。ここで PDF の `PDDoc` オブジェクトを開き、その文書に対応する `JSObject` インタフェースを取得します。

また、標準の Acrobat オートメーションメソッドをいくつか使用して、文書の先頭ページのサイズを取得しています。PDF の座標系はページの左下隅を基点としていますが、注釈はページの左上隅を基点とするため、ここで取得した数値は正しいレイアウトを実現する上で重要となります。

「Create a new text annot」というコメントに続く行では注釈の作成が行われますが、このコードブロックにはそれ以外にも注目すべき点があります。まず最初に、`addAnnot` は `JSObject` のメソッドのように見えますが、JavaScript リファレンスを見ると、このメソッドは `Doc` オブジェクトに関連付けられています。この場合、`jso.doc.addAnnot` という構文が正しいと思われるかもしれませんが、`jso` は `Doc` オブジェクトであるため、`jso.addAnnot` が正しい構文になります。`Doc` オブジェクトのプロパティおよびメソッドはすべて、このようにして使用できます。

次に注目すべき点は、`annot.getProps` と `annot.setProps` の使用方法です。`Annot` オブジェクトは、別個のプロパティオブジェクトを使用して実装されているため、プロパティを直接設定することはできません。例えば、次のような設定は行えません。

```
Set annot = jso.AddAnnot
annot.Type = "Text"
annot.page = 0
...
```

この場合は、`annot.getProps` を使用して `Annot` のプロパティオブジェクトを取得し、そのオブジェクトを使用して読み取り／書き込みを行う必要があります。変更内容を元の `Annot` に保存するには、先ほどの例と同様に、変更済みのプロパティオブジェクトを指定して `annot.setProps` を呼び出します。

最後に、`JSObject` のカラープロパティの使用法に注目してください。このオブジェクトではいくつかの単色（赤、緑、青など）が定義されていますが、このオブジェクトで使用可能な色とは異なる色を使用したい場合もあります。また、`JSObject` を呼び出すと、そのぶんパフォーマンスが落ちます。色をより効率的に設定するには、次のようなコードを使用できます。このコードでは `color` オブジェクトを省略し、直接 `annot.strokeColor` を赤に設定しています。

```
dim color(0 to 3) as Variant
color(0) = "RGB"
color(1) = 1#
color(2) = 0#
color(3) = 0#
annot.strokeColor = color
```

この方法は、`JSObject` の関数のパラメータとしてカラー配列が必要となるすべての状況で使用できます。この例では、カラー空間を RGB に設定し、0～1 の浮動小数点値を赤、緑、および青として指定しています。カラー値の後に # 文字が使用されていることに注目してください。この文字は、Visual Basic に対して、配列要素を整数ではなく浮動小数点値に設定するように指示するためのものであり、必須の文字です。また、この配列には文字列と浮動小数点値の両方が含まれるので、`Variant` の配列として宣言することが重要です。その他のカラー空間（「T」、「G」、「CMYK」）では、配列長に関する要件がそれぞれ異なります。詳しくは、『[Acrobat JavaScript Scripting Reference](#)』で `color` オブジェクトの説明を参照してください。

文書のスペルチェック

Acrobat 6.0 には、文書内のスペルミスを検出できるプラグインが組み込まれています。このプラグインには、JObject を使用してアクセスできる JavaScript メソッドも用意されています。この例では、「例 2」のソースコードを基にして、次の変更を加えます。

1. メインフォームに ListBox コントロールを追加します。このコントロールの名前にはデフォルト値 (`List1`) をそのまま使用します。
2. 既存の `Command1_Click` 関数のコードを次のコードに置き換えます。

例 3 文書のスペルチェック

```
Private Sub Command1_Click()
    Dim pdDoc As Acrobat.CAcroPDDoc
    Dim jso As Object
    Dim path As String
    Dim count As Long
    Dim i As Long, j As Long
    Dim word As Variant
    Dim result As Variant
    Dim foundErr As Boolean

    CommonDialog1.ShowOpen
    path = CommonDialog1.FileName
    foundErr = False
    Set pdDoc = CreateObject("AcroExch.PDDoc")

    If pdDoc.Open(path) Then
        Set jso = pdDoc.GetJSObject
        If Not jso Is Nothing Then
            count = jso.getPageNumWords(0)
            For i = 0 To count - 1
                word = jso.getPageNthWord(0, i)
                If VarType(word) = vbString Then
                    result = jso.spell.checkWord(word)
                    If IsArray(result) Then
                        foundErr = True
                        List1.AddItem word & " is misspelled."
                        List1.AddItem "Suggestions:"
                        For j = LBound(result) To
UBound(result)
                            List1.AddItem result(j)
                        Next j
                        List1.AddItem ""
                    End If
                End If
            Next i
            Set jso = Nothing
            pdDoc.Close

            If Not foundErr Then
                List1.AddItem "No spelling errors found in " & path
            End If
        End If
    End Sub
```

```
                End If
            End If
        Else
            MsgBox "Failed to open " & path
        End If

        Set pdDoc = Nothing
    End Sub
```

この例では、Spell オブジェクトの `checkWord` メソッドの使用方法に注目してください。『[Acrobat JavaScript Scripting Reference](#)』によると、このメソッドは単語を入力として取り、その単語が辞書に存在する場合はヌルオブジェクトを返し、その単語が見つからない場合は修正候補の配列を返します。JSObject メソッド呼び出しの戻り値を格納するのに最も安全な方法は、Variant を使用することです。IsArray 関数を使用すると、Variant が配列であるかどうかを確認して、適切に操作することができます。この例では、修正候補の配列が返された場合、ListBox コントロールにダンプします。

JavaScript を JScript に変換する際のヒント

JavaScript を JScript に変換する際のヒント

このマニュアルでは、JScript で使用可能なメソッドの一部しか取り上げていません。詳しくは、『[Acrobat JavaScript Scripting Reference](#)』を参照してください。次に示す基本的な事実を念頭に置きながらこのリファレンスを読むと、一層理解が深まります。

1. このリファレンスで説明されているオブジェクトおよびメソッドの大半は Visual Basic で使用できますが、すべてを使用できるわけではありません。特に、生成に **new** 演算子が必要な JavaScript オブジェクトは、Visual Basic では作成できません。これには **Report** オブジェクトが含まれます。
2. **Annots** オブジェクトは、JScript の **getProps** および **setProps** メソッドを使用してプロパティを別個のオブジェクトとして設定したり取得したりする必要があるという点で、特殊なオブジェクトです。
3. 変数の宣言で使用すべき型が不明確な場合は、Variant として宣言します。これにより、Visual Basic の型変換を柔軟に行えるようになり、実行時のエラーを回避できます。
4. JScript は、新しいプロパティ、メソッド、またはオブジェクトを JavaScript に追加できません。したがって、**global.setPersistent** プロパティは意味を持ちません。
5. JScript では、大文字と小文字が区別されません。Visual Basic では、多くの場合、識別子の頭文字に大文字が使用され、大文字/小文字を変更することができません。しかし、JScript は識別子を JavaScript と対応させる際に大文字と小文字を無視するので、この点を気にする必要はありません。
6. JScript は、常に値を Variant として返します。これには、メソッド呼び出しからの戻り値のほかに、プロパティの取得も含まれます。ヌルの戻り値が想定されるときは、空の Variant が使用されます。JScript が配列を返すときは、配列の各要素が Variant になります。Variant の実際のデータ型を確認するには、VBA ライブラリの Information モジュールに含まれる **isArray**、**isNumeric**、**isEmpty**、**isObject**、および **varType** ユーティリティ関数を使用します。
7. JScript は、プロパティの設定やメソッド呼び出しの入力パラメータとして、Visual Basic の基本的な型のほとんど (Variant、Array、Boolean、String、Date、Double、Long、Integer、Byte など) を扱うことができます。JScript は Object パラメータを受け取ることができますが、ただし、その Object がプロパティの取得の結果であるか、または JScript に対するメソッド呼び出しの結果である場合に限られます。JScript は、Error 型および Currency 型の値は受け取れません。