PowerPoint AppleScript

Microsoft® PowerPoint®:mac 2008

このドキュメントに記載されている情報 (URL などのインターネット Web サイトに関する情報を含む) は、情報提供の目的で発行されているものであり、将来予告なしに変更することがあります。

別途マイクロソフトのライセンス契約上に明示の規定のない限り、このドキュメントはこれらの特許、商標、著作権、またはその他の無体財産権に関する権利をお客様に許諾するものではありません。

Microsoft、Excel、Entourage、PowerPoint は、米国 Microsoft Corporation およびその関連会社の登録商標または商標です。

Apple、Apple ロゴ、Mac、Mac OS は、米国 Apple Inc. 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。

PowerPoint AppleScript

PowerPoint AppleScript の基本

Microsoft PowerPoint 2008 では Visual Basic for Applications (VBA) がサポートされていないため、以前のバージョンで作成された VBA コードをそのまま使用することはできませんが、AppleScript に変換することで最新バージョンの PowerPoint でも利用できます。

PowerPoint AppleScript トピックでは、VBA で一般的によく使用されるサブルーチン (マクロ) と、これに相当する AppleScript サブルーチンについて説明します。

Macintosh スクリプト エディタ でスクリプトをテストするときは、各スクリプトの `tell application "Microsoft PowerPoint"` ブロックの先頭に単独行として `activate` コマンドを指定することができます。これにより、PowerPoint が前面に表示されるため、動作を確認しやすくなります。PowerPoint が前面に表示されているときに、保存したスクリプトを [スクリプト] メニューから実行する場合は、`activate` を指定する必要はありません。他の場所からスクリプトを実行していて、PowerPoint を前面に表示する必要がない場合も同様です。各トピックで紹介するサンプル スクリプトには `activate` コマンドは含まれていませんが、必要に応じて使用することもできます。

PowerPoint でスクリプトの作成を開始する前に、PowerPoint 2004 AppleScript リファレンス (http://download.microsoft.com/download/2/4/C/24CADB5A-88A9-4A6C-8100-8F5DCF65C21C/PPT2004AppleScriptRef_J.pdf) の「PowerPoint Dictionary の使用」をお読みになることをお勧めします。このドキュメントは、AppleScript よりも PowerPoint オブジェクト モデルに慣れていないユーザーを対象にしていますが、すべての PowerPoint スクリプトの作成者に役立つ情報が記載されています。

text frame を使用してスクリプトを記述する場合は、Word AppleScript のトピックで **text range** に関する情報をお読みください。また、Word 2004 AppleScript リファレンス (http://download.microsoft.com/download/A/0/7/A07D8BDE-8903-4A78-BD84-E2CF6A9DCF33/Word2004AppleScriptRef_J.pdf) の導入部分も参照してください。注意すべき点は、Dictionary の Drawing Suite に含まれる **text frame** は、PowerPoint、Microsoft Word のどちらの場合も、範囲プロパティとして **text object** ではなく **text range** を使用するクラスであることです (Text Suite には、同じ名前の **text range** クラスがあります)。 詳細については、「検索と置換」を参照してください。

PowerPoint 2004 AppleScript を使用していて、スクリプトの変換中に問題が発生して回避策がわからない場合は、**do Visual Basic** コマンドで 1 行または複数行を置換します。つまり、記述された VBA の行を引用符で囲んでテキストにします (Microsoft Visual Basic Editor の環境設定で [変数の宣言を強制する] をオンにしている場合でも、これらに対して `Dim` 宣言は不要です。ただし必要に応じて含めることもできます)。従来は、PowerPoint X のスクリプトを作成するには **do Visual Basic** コマンドを使用する以外に方法はありませんでした。このコマンドは PowerPoint 2004 でも問題なく動作しますが、PowerPoint 2008 では動作しません。VBA の文字列内を引用符で囲む場合は、すべて \ (円記号またはバックスラッシュ) を付けてエスケープする必要があります。

PowerPoint 2004 AppleScript の **application** および **presentation** には、**selection** オブジェクトがありません。一方、VBA マクロでは **Selection** オブジェクトを参照している場合が数多くあります。そのため、**do Visual Basic** コマンドを使用して **Selection** オブジェクトを取得 (`Selection.ShapeRange` と `Selection.SlideRange` によって特定のスライドと図形を取得) することはできますが、**do Visual Basic** コマンドでは、**result** (選択したスライドまたは図形) を AppleScript に返すことができません。このような場合は、`.Selection` を使用しないでマクロを書き直すことができるかどうかを確認してください。

プレゼンテーションを作成するまたは開く

プレゼンテーションを作成する

Visual Basic for Applications (VBA) で新しい空白の **presentation** を作成するには、次のコードを使用します。

```
Application.Presentations.Add
```

AppleScript では次のようになります。

```
make new presentation
```

Microsoft PowerPoint では、**make new** コマンドを使用してプレゼンテーションを作成する際、プロパティを設定することはできません。AppleScript では通常、ドキュメントの作成時に、**with properties** パラメータにドキュメントのプロパティの大半を指定できます (スクリプト作成者は通常、必要な要素だけを後で追加することになります)。しかし、新しいプレゼンテーションの作成時にはこれができません。プロパティを設定するには、先にプレゼンテーションを作成する必要があります。これは、**Presentation** に対する **Add** メソッドの動作と同じであるため、VBA に慣れているユーザーにとって目新しいことではありません。

VBA で、`Presentations.Add` に指定できる引数は **WithWindow** のみです。これは新しいプレゼンテーションが **visible** であるかどうかを決定する引数です。AppleScript には、プレゼンテーションを非表示にする方法がありません。プレゼンテーションには **visible** プロパティがなく、*false* に設定することができません (メインの **document window** には **visible** プロパティがありますが、多くのアプリケーションと同様、読み取り専用で、常に *true* に設定されます)。また、`hide` のようなコマンドもありません。

一見すると AppleScript で設定できるプレゼンテーションのプロパティはないように思われます (**presentation** 向けの Dictionary エントリでは、すべてのプロパティが `r/o` (読み取り専用) となっています)。しかし、**page setup**、**web options**、**slide show settings** など、数多くのプロパティはオブジェクトを返し、そのオブジェクトのプロパティをほぼ無制限に設定できます。これは、各 Microsoft Office アプリケーションでも同じです。つまり、プレゼンテーションの作成後であればこうした "プロパティのプロパティ" を設定できますが、**make new** ステートメントでは設定できないということです。また、プレゼンテーションの作成後に作成できる新しい要素もあります。これらの要素は、後から必要に応じて変更 (つまり、要素独自のプロパティを設定) できます。

VBA でプロパティと見なされるものには多数のコレクション オブジェクト (**Slides** など) が含まれますが、これは AppleScript ではスクリプト作成者が作成する要素です。VBA でも AppleScript でも、**presentation** クラスのプロパティは、**east asian line break level** および **layout direction** (PowerPoint 英語 (US) 版では未提供) のようないくつかのプロパティを除き、すべて読み取り専用です。

また、AppleScript には、読み取り/書き込みプロパティの **DisplayComments** プロパティに相当するものがありません。Drawing Suite でコメントを作成し、それに対してあらゆる種類の属性を設定できますが、表示するかどうかを制御することはできません。

以上のことから、新しいプレゼンテーションの作成が必要なスクリプトはすべて、次の記述から始まります。

```
tell application "Microsoft PowerPoint"
        set newPres to make new presentation
        tell slide 1 of newPres
                -- code here
        end tell
end tell
```

`make new presentation` ステートメントには、任意の変数を設定する必要があります。この定義では、作成したオブジェクトを返すようにします。このようにしないと、操作対象がなくなります。常に `slide 1` の各種のプロパティを設定したり `tell slide 1` で新しい要素を作成するわけではありませんが、ほとんどの場合はこれを指定するのが最も手っ取り早い回避策です。

既存のプレゼンテーションを開く

AppleScript では、既存の PowerPoint プレゼンテーション ファイルを簡単に開くことができます。VBA の **Open** メソッドを次に示します。

```
Presentations.Open FileName:="Mac HD:Folder:Filename.ppt"
```

AppleScript でこれに相当するものは、次に示す Standard Suite の **open** コマンドです。

```
tell application "Microsoft PowerPoint"
        open alias "Mac HD:Folder:Filename.ppt"
end tell
```

エイリアス参照には適切な形式を使用する必要がありますが、PowerPoint では強制型指定が可能になっています。このため、VBA と同じく **path** テキストだけを使用して同じものを指定できます。また、AppleScript では、必要に応じてプレゼンテーション ファイルの一覧を同時に開くことができます。それには .ppt エイリアスの一覧を指定するだけです。

また、プレゼンテーションを変更したり、プレゼンテーションから情報を取得したりすることもできます。VBA では、変数 (参照) を **Open** ステートメントの結果に設定し、それによって **Presentation** オブジェクトを返します。その後、そのオブジェクトのプロパティを設定するか、そのオブジェクトにメソッドを適用します。

```
Dim oPres As Presentation, oSlide As Slide

        Set oPres = Presentations.Open(Mac HD:Folder:Filename.ppt")
```

```
        With oPres
                Set oSlide = .Slides.Add(2, ppLayoutText)
        End With
```

変数をコマンドの結果に設定する手順は通常、AppleScript でも簡単です。ただし、Microsoft Word と同じように、Standard Suite のコマンド **open** では結果が返されないため、このコマンドには変数を設定できません。

次のようにすることもできます。

```
tell application "Microsoft PowerPoint"
        open "Mac HD:Users:yourname:Desktop:Saved Pres.ppt"
        set thePres to active presentation
        set theSlide to make new slide at end of thePres¬
                with properties {layout:slide layout text slide}
end tell
```

ここでは、プレゼンテーションを開き、変数を **active presentation** (前面にあるプレゼンテーション) に設定します。**open** コマンドは常に **application** の前面にあるファイルを開きます。したがって、この方法はプレゼンテーションが開かれていない場合に有効です。逆にプレゼンテーションが開かれている場合は前面に表示されず、誤ったプレゼンテーションに対して実行することになります。必要なコードは全体で次のようになります。

```
tell application "Microsoft PowerPoint"
        try
                set thePres to presentation "Saved Pres.ppt"
        on error
                open "Mac HD:Users:yourname:Desktop:Saved Pres.ppt"
                set thePres to active presentation
        end try
        set theSlide to make new slide at end of thePres ¬
                with properties {layout:slide layout text slide}
end tell
```

スライドと図形を使用する

スライドを作成するまたは挿入する

スライドを作成する

Visual Basic for Applications (VBA) の **Slides** の **Add** メソッドには、**Layout** および **Index** の 2 つの引数が必要です。

```
ActivePresentation.Slides.Add Index:=1, Layout:=ppLayoutText
```

AppleScript には、**slide** の読み取り/書き込みのプロパティがそれほど多く存在しません。また、**color scheme** のような一部のプロパティは、変更の際に特殊な独自コマンドを必要とするため、新しいスライドを作成する際に最初から設定することができません。たとえば、**layout** プロパティは設定できますが、**Index** に対応する AppleScript の **slide index** プロパティは読み取り専用で設定できず、一部の読み取り専用のプロパティのように作成時に設定することもできません。

しかし、AppleScript には要素の挿入場所を指定する方法があります。通常は、**at** を使用してオブジェクトに要素を作成しますが、次の記述ではエラーが発生します。

```
tell application "Microsoft PowerPoint"
        set newSlide to make new slide at active presentation ¬
                with properties {layout:slide layout text slide}
end tell
```

Dictionary の Standard Suite で **make** コマンドを参照すると、**new** と **at** の 2 つのパラメータが必要であることがわかります。内部的には、この **new** 引数が新しい要素のクラスを指定します。**at** パラメータの説明には、`location reference : the location at which to insert the element` とあります。厳密には、単純に親オブジェクトを指定する代わりに、正確な位置を指定する必要があります。ところが実際は、新しい要素の挿入位置が問題でなかったり、最後 (既存要素の後ろ) に配置する以外の選択肢がないことがあるため、`at end of [object]` ステートメントを省略することができます。

しかし、**presentation** のスライドでは、最後だけではなく任意の場所に挿入することができます。AppleScript には挿入位置を示す方法として、`at beginning of`、`at end of`、`at before [some element]`、および `at after [some element]` があります。

```
tell application "Microsoft PowerPoint"
        set newSlideA to make new slide at beginning of active presentation ¬
                with properties {layout:slide layout text slide}
        set newSlideB to make new slide at end of active presentation ¬
                with properties {layout:slide layout two objects over text}
        set newSlideC to make new slide at before slide 2 ¬
                of active presentation with properties ¬
                {layout:slide layout media clip and text}
end tell
```

最後のコマンドでは、既存の `slide 2` の前に新しい `slide 2` として `newSlideC` を挿入します。これと同じ位置に挿入するのに、`at after slide 1` を使用することもできます。`before` の前に **at** を配置することは英語としては正しくありませんが、必ず必要なので注意してください。スライドを作成したら、その **slide index** プロパティをいつでも取得して配置場所を確認することができます。

レイアウトの種類を確認するには、Dictionary の Microsoft PowerPoint Suite の **slide** クラスに記載されている **layout** プロパティの列挙定数を参照します。たとえば、VBA の定数 **ppLayoutText** が **slide layout text slide** に相当することは容易にわかります。

メモ これまで、開発者がコードを記述する際には、用語の重複が生じないよう細心の注意が必要でした。Dictionary やスクリプト内のキーワードでは、実際にコンパイルされる "生のコード" が隠されているので、同じキーワードが異なるプロパティや列挙体に使用されると、スクリプトは正しくコンパイルされず失敗します。このような重複をなくす最も簡単な方法は、**slide layout** のように、クラスとプロパティから始まって列挙定数が続く一意の名前を列挙体に付けることでした。たとえば、列挙定数で頻繁に使用される `text` を 3 番目に持つものは 6 個以上ありますが、他の列挙と混同されないように `slide` が追加されています。VBA の定数に対応する AppleScript 用語は、VBA の定数の前に **slide layout** を付けた形に似た名前を探すことで簡単に見つかります。

説明的な定数 `pp` ではなく **Long** (数値) がそのまま使用されているマクロをときどき見かけます。たとえば **ppLayoutText** ではなく `2` が使用されている場合などです。困ったことに、これらは Microsoft Office のいくつかの列挙体では有効ですが、この場合は使用できません。Microsoft Visual Basic Editor のオブジェクト ブラウザで **ppSlideLayout** を調べ、Long (`2`) を基にその名前 (**ppLayoutText**) を探し、AppleScript の列挙リストから相当するものを見つけて使用する必要があります。

スライドを作成したら、追跡できるようにしておく必要があります。スライドへの参照は正式にはインデックス (`slide 1`、`slide 2`、`slide 3` など) を使用しますが、先頭または途中に新たなスライドを挿入するとインデックスは変更されます。たとえば、`slide 3` と認識していたスライドが `slide 3` ではなくなり、`slide 5` になります。

VBA では、すべてのスライドに **Name** プロパティがあります。既定の名前は `Slide1`、`Slide2` などですが、変更することもできます。AppleScript には **name** プロパティはありません。しかし、すべてのスライドには一意の ID (VBA では **SlideID** プロパティ) もあり、スライド作成後にすぐに取得できます (PowerPoint を初めて起動したときから、ID が増えていくように見えます)。AppleScript では、**slide ID** プロパティとして、このプロパティが存在します。VBA では、**Slides Collection** オブジェクトの **FindBySlideID** メソッドで SlideID を使用し、スライドを見つけ出すこともできます。AppleScript にはそのようなコマンドがありません。

ここで AppleScript の `whose` フィルタに注意してください。次のコードは有効であるように見えますが、結果は空のリストの { } であり、中身はありません。

```
get first slide of active presentation whose slide ID is 1257
```

式 `first element whose...` は Office アプリケーション間で統一されていないため、正しく実行される場合もあれば、`every element whose...` と同じ結果になることもあります。ここでは、代わりに次のようにします。

```
get every slide of active presentation whose slide ID is 1257
```

これによって、リストの中かっこ内に正しい結果が得られます。

```
--> {slide 1 of active presentation}
```

該当する ID を持つスライドは当然 1 つしかないため、`every` で取得されるのは単一項目から成るリストです。そこで、次のように記述します。

```
item 1 of (get every slide of active presentation whose slide ID is 1257)
        --> slide 1 of active presentation
```

スライドを挿入する

ファイルに保存した既存のスライドをプレゼンテーションに挿入するには、VBA では **InsertFromFile** メソッドを使用しますが、AppleScript にはそれに相当するものがありません。そこで **do Visual Basic** を使用します。マクロの変換は可能であり、たとえば次のような InsertFromFile を使用する行があるとします。

```
ActivePresentation.Slides.InsertFromFile _
        "Macintosh HD:Users:Shared:Sales.ppt", 2, 3, 6
```

ここでは、次のコードを使用できます。

```
do Visual Basic "ActivePresentation.Slides.InsertFromFile _
        \"Macintosh HD:Users:Shared:Sales.ppt\", 2, 3, 6"
```

内側の引用符の前のエスケープ用の \ (円記号またはバックスラッシュ) と、VBA から引用している各行を囲む外側の AppleScript の引用符を閉じる必要がないことに注意してください。また、この行は Office 2008 ではエラーになります。

スライドまたは図形を選択する

Visual Basic for Applications (VBA) では以下のようなコードをよく見かけます。

```
Dim oSh as Object
Set oSh = ActivePresentation.Slides(4).Shapes(8)
With oSh
        .SomeProperty = "this"
        .AnotherProperty = "that"
End With
```

AppleScript では、このような宣言 (`Dim`) は不要 (不可能) です。また、AppleScript にはコレクション オブジェクトがないため、`ActivePresentation.Slides(4).Shapes(8)` の代わりに `shape 8 of slide 4 of active presentation` を参照します。 さらに、次に示すように VBA での `With` ブロックの代わりに、`tell` ブロックを同様に使用できます。

```
tell application "Microsoft PowerPoint"
        set theShape to shape 8 of slide 4 of active presentation
        tell theShape
                set some property to "this"
                set another property to "that"
        end tell
end tell
```

また、`theShape` プロパティを 18 個程度設定する場合は、各行で 1 つずつ設定する必要はありません。必要であれば、各行に分けることもできます (そうした方が追跡しやすいこともあります)。以下のように、リストを使用して長い 1 行ですべてを設定できます。

```
tell theShape
        set {some property, another property} to {"this", "that"}
end tell
```

1 行でプロパティのリストを設定する場合は、tell ブロックを使用する必要があります。of 構文を使用して同様の記述を行うことはできません。

AppleScript には、**slide** の **name** プロパティがありません。しかし、VBA でスライドの追跡と参照を行うにはこのプロパティが便利なため、変換しようとする多くのマクロでは、各スライドと対象の **shape** に名前が付けられ、後で参照するようになっています。多くのマクロは、スライドまたは図形が選択済みであることを前提に始まり、後の参照のための名前付けに進んでいます。

```
With ActiveWindow.Selection.SlideRange
        .Name = "MyName"
End With

With ActiveWindow.Selection.ShapeRange
        .Name = "MyName"
End With
```

VBA 作成者は、上に示すような **SlideRange** または **ShapeRange** を使用するか、.Selection.Slides(n) を使用して、**Name** プロパティで 1 つのスライドまたは図形のみを選択するように指定していることが考えられます。

このコード形式は AppleScript では使用できません。Microsoft PowerPoint 2004 for Mac 用の AppleScript には、**selection** オブジェクトがありません。選択されているスライドを取得する代わりの方法としては、最初にスライド ウインドウ内の任意の場所をクリックしてアクティブ ウインドウにします。

```
tell application "Microsoft PowerPoint"
        set theIndex to slide index of slide of view ¬
                of active window
        set selectedSlide to slide theIndex of active presentation
        set slideID to slide ID of selectedSlide
end tell
```

スライド一覧表示ではなく標準表示で、スライド ウインドウ内の任意の場所を最初にクリックしてアクティブにすることで、**active window** の **view** にあるスライドが選択スライドになります。スライド一覧表示では、どのスライドも **slide** プロパティとしては表示されません。または、スライド表示に切り替えると常にうまくいきます。スライドには名前を付けることができないので、代わりに **slide ID** を使用します。

別のスライド位置に移動しない場合は、スクリプトで selectedSlide を後で呼び出すだけで参照できます。ただし、これは **slide index** への参照であり、インデックスは変化することがよくあります。スライド ID をすぐに取得し (上に示すコードの 3 行目)、その後取得した ID を使用してスライドを参照する方が安全です。

```
set theSlide to item 1 of (get every slide of active presentation ¬
        whose slide ID is slideID)
```

スライド上に複数の図形がある場合に、特定の選択図形を取得することは難しく感じられます。**do Visual Basic** コマンドは AppleScript に結果を返さないため、一見機能しないように見えます。

この 1 行目を無視し、マクロを AppleScript に変換できるようにする回避策があります。この方法では、GetSelectedShape() と呼ばれるハンドラ (サブルーチン) を使用します。また、これは今後頻繁に必要になります。スクリプト ライブラリで GetSelectedShape() をすぐに使用できるようにしておき、選択した図形を扱う必要がある各スクリプトの最後にペーストしてください。

選択されている図形を取得したり ActiveWindow.Selection.ShapeRange へ変数を割り当てたりする VBA 行の置き換えが必要な場合はいつでも、次の行を入力してハンドラを呼び出します。

```
set selectedShape to my GetSelectedShape()
```

次に実際の使用例を示します。

```
tell application "Microsoft PowerPoint"
        set selectedShape to my GetSelectedShape() --calls the handler below
        -- test only:
        set theText to (content of text range of text frame of selectedShape)
        -- enter rest of script after trying the test and removing the line above
end tell

to GetSelectedShape()
        set textFilePath to (path to temporary items as string) & "Shared Text File"
        tell application "Microsoft PowerPoint"
             activate
             set theIndex to slide index of slide of view ¬
                       of active window
             set selectedSlide to slide theIndex of active presentation
             do Visual Basic "n = ActiveWindow.Selection.ShapeRange.ZOrderPosition
        'write n as string to text file, replacing any text there
        FileNumber = FreeFile
        Open \"" & textFilePath & "\" For Output As #FileNumber
        Print #FileNumber, CStr(n)
        Close #FileNumber
        "
             set n to (read alias textFilePath before return) as integer
             set selectedShape to shape n of selectedSlide
             return selectedShape
        end tell
end GetSelectedShape
```

最上位のスクリプトの最終行でテキストを取得していますが、これはスクリプトをすぐにテストできるようにするためのものです。このスクリプトを実行する前には、テキストが含まれる **text box** または図形を選択してください。Macintosh スクリプト エディタ に、スクリプトの結果 (選択した図形のテキスト) が表示されます。これは選択した図形のテキストです。

AppleScript では、前の例と同じようにして、最初にビューで選択スライドを取得します。この場合、図形を選択しているので、スライド ウインドウが確実にアクティブになっています。**do Visual Basic** を使用して、選択した図形の **ZOrderPosition** を取得します。図形の **ZOrderPosition** プロパティ (AppleScript では **z order position**) は、"新しい順"、つまり作成順の逆を示すインデックス番号です。UI、マクロ、またはスクリプトで新しく図形を作成するたびに、その図形に `z order position 1` が割り当てられ、他の図形は大きな数に繰り上げられます。このインデックス番号 `n` はスライド上での **shape index** と同じです。選択されているスライドはわかっているので、そのスライドの `shape n` を取得できます。これが、選択されている図形です。

ここでの問題は、**do Visual Basic** コマンドが `n` をスクリプトに返すことができないということです。この問題を解決するため、代わりに整数値 `n` を文字列 (`CStr(n)`) にキャストしてテキスト ファイルに書き出します。ハンドラの最初の行では、Macintosh の [Temporary Items] フォルダ内に事前に準備したファイルへのパスを記述し、変数 `textFilePath` をそのパスに設定しています。この `textFilePath` 変数を、エスケープした引用符で囲んで **do Visual Basic** コマンドに挿入します。次に、標準の Visual Basic メソッドである **FreeFile**、**Print #**、**Close #** を使用して、文字列である数値をファイルに書き込み、ファイルを閉じます。

メモ Mac OS X では、[Temporary Items] フォルダはユーザー フォルダではなく、ハード ドライブのルートの "/var/tmp" にあります。そのため、使用しているコンピュータに他のユーザーがアクセスする可能性がある場合は、重要なデータを書き込まないようにしてください。または、スクリプトの最後で Finder がそのファイルを削除するように指定してください。ただし、今回書き込むのは `3` や `4` という数字であり、特に注意する必要はありません。

このようにすると、AppleScript の **read Standard Addition** コマンドを使用してファイルから文字列 (Print # で追加された改行記号まで) を迅速に取得できます。これを整数値 `n` に戻して、`shape n of selectedSlide` を取得することで、選択した図形を得ることができます。

ここに示した回避策は Microsoft Office 2004 でのマクロの変換に使用できますが、Office 2008 では **do Visual Basic** コマンドは機能しません。ハンドラによって、最初の問題が解決され、選択した図形に依存するマクロを変換できるようになります。

スライドと図形の書式を設定する

slide 上のすべての **shape** と、**presentation** 内のすべてのスライドを変更できます。

プレゼンテーション内のすべてのスライドの背景の **color** を淡いアクアマリンに設定するマクロを次に示します。この背景色は、 [書式] メニューの [スライドの配色] をクリックして、2 行目の先頭にある配色のものです。次に、各スライド上の各図形を 0.5 インチ右に移動します。この処理を、プレゼンテーション内の各スライドとスライド上の各図形に対してループします。

```
Dim oPres As Presentation
Dim oSl As Slide
Dim oSh As Shape
Dim colSch As ColorScheme

' Get a reference to the current active presentation
Set oPres = ActivePresentation
```

```
' Do something with each slide in the presentation
For Each oSl In oPres.Slides
        ' Set the background color scheme of each slide to a pale aquamarine
        Set colSch = oSl.ColorScheme
        colSch.Colors(ppBackground).RGB = RGB(222, 246, 241)
        ' and do something with each shape on each slide
        For Each oSh In oSl.Shapes
                ' move it 36 points (.5 inch) to the right
                oSh.Left = oSh.Left + 36
        Next
Next
```

同じ処理を AppleScript で記述すると次のようになります。

```
tell application "Microsoft PowerPoint"
        set oPres to active presentation
        repeat with oSl in (get every slide of oPres)
                set color for (color scheme of oSl) at background scheme ¬
                        to color {222, 246, 241}
                repeat with oSh in (get every shape of oSl)
                        tell oSh
                                set left position to (get left position) + 36
                        end tell
                end repeat
        end repeat
end tell
```

`Dim` 宣言は省略し、わかりやすくするために VBA と同じ変数を使用します。ただし、object、string、long などの型を表す変数の頭文字 `o`、`s`、`l` などを使用する必要はありません。AppleScript ではこれらは区別されません。数か月または数年後にスクリプトを参照することがある場合は、`theSlide` や `theShape` のような説明的な用語を使用してスクリプトを記述しておくと、スクリプト自体がコメントの役割を果たすので多くの追加コメントが不要になり、内容の確認が簡単になります。

このマクロは簡単に AppleScript に変換することができます。**color scheme** クラスは、標準的な方法で取得したり設定したりするプロパティ (**background scheme color**、**fill scheme color** など) が一切ないという点で通常のクラスと異なっています。その代わりに、独自の **get color for** コマンドと **set color for** コマンドを使用する必要があります。これらのコマンドの **at** パラメータは同じ定数の列挙体にアクセスします (これはおそらく、**ColorScheme.Colors** がプロパティというよりメソッドであり、列挙定数 `pp` を引数として持つ VBA の構造を継承したことによるものです)。ここでは定数 *background scheme* を使用します。

色の値については少し説明が必要です。VBA の **RGBColor** オブジェクトは、Red-Green-Blue を表す 3 つの 8 ビット整数値 (0 ～ 255) の配列である **RGB** プロパティから成る **Long** 値です。Microsoft Office AppleScript (ここでは Word、Excel、および PowerPoint のみを対象としています) の、型が **color** で表されるプロパティまたはクラスには、同じ 3 つの整数値 (0 ～ 255) をリストとして中かっこで囲み、`{222, 246, 241}` のように指定します。これは "Microsoft color クラス" の一種であり、VBA と同じ RGB の 3 つの数値を使用できるので、VBA マクロを AppleScript に変換する際はきわめて便利です。

しかし、Apple の定義による AppleScript 本来の RGB color クラスは、3 つの 16 ビット整数値 (0 ～ 65535) から成る RGB のリストです。Word、Excel または PowerPoint から、Microsoft Entourage (このカテゴリには **color** プロパティがあります) を含む別の Macintosh アプリケーションに色を渡す場合は、同じ 3 つの数字を使用するとまったく違う色になります。同じ色にするには、次のサブルーチンにあるように各整数値を 2 乗します。

```
ApplifyMSColor({222, 246, 241})
--> {49284, 60516, 58081}

to ApplifyMSColor({r, g, b})
        set r to (r ^ 2) as integer
        set g to (g ^ 2) as integer
        set b to (b ^ 2) as integer
        return {r, g, b}
end ApplifyMSColor
```

`as integer` が必要である理由は、AppleScript の指数演算子 ^ が **reals** (浮動小数点数) を生成するためです。浮動小数点数は、たとえば 10000.0 以上の場合は 4.9284E+4 のような形式で表されます。最大値 65535 であるこれらの値を整数型に強制型変換を行い、color 型で使用できるようにします。逆に、Entourage から PowerPoint に色を渡す場合は、各整数値 (65535 ベース) の平方根を取得して、最も近い整数値に切り捨てます。これは次に示すように、整数の除算演算子 `div` によって簡単に実行できます。

```
set r to (49285 ^ 0.5) div 1
-->222
```

強制型変換 `as integer` は Mac OS X バージョン 10.4 (Tiger) 以降では使用できますが、それ以前のオペレーティング システムでは使用できません。65535 ベースの数値が完全な 2 乗ではなかった場合、その平方根は小数点以下がゼロではない実数になります。たとえば 499285 ^ 0.5 の結果は 222.002252240828 になります。Mac OS X バージョン 10.4 以前のバージョンでは、そのような実数に対して `as integer` の強制型変換を試みるとエラーが発生します。このため、Mac OS X の全バージョンで有効な `div 1` を使用します。

図形の **left position** プロパティと VBA の **Left Property** は完全に対応しているため、図形の移動は簡単です。別の行で先にプロパティ (**left position**) を変数に設定するのではなく、複合的なステートメントの中でプロパティの取得とそれに対する処理を同じ行で実行する場合は、取得に対して **get** を明示的に使用する必要があります。

```
set left position to (get left position) + 36
```

また、`repeat` ループ形式である `repeat with someVariable in someList` の中でも、前もって変数 (`someList`) を設定するのではなく `every element of someObject` をリストとして使用する場合は、明示的に **get** コマンドを使用する必要があります。

VBA プログラム作成者の多くは VBA での記述方式に合わせて AppleScript を記述する傾向にあり、ここで **get** コマンドを明示的に使用するのはそのためですが、最初に変数を設定し、式を評価してから操作を行う方法をお勧めします。**get** コマンドを明示的に使用することは忘れやすく、スクリプトでエラー `can't get [whatever]` が発生し続ける理由がわからなくなることがよくあります。

スライド上のすべての図形の左位置を、それぞれの元の位置からの相対位置ではなく同じ値に設定する場合、AppleScript では repeat ループを使用しないで一度に実行できます。

```
set left position of every shape of oSl to 100
```

繰り返し処理が不要になるので、たとえばスライド上に図形が 50 もあるような場合にきわめて便利です。every element 構文はオブジェクト専用に実装する必要があるため、その使用は常に有効であるとは限りませんが、ここでは有効です。プロパティを設定するのではなく独自コマンド **set color for** を呼び出しているので、配色の背景の設定には使用できません。

テキストを使用する

テキストを選択、追加、変更する

テキストを入力する場所がテキスト ボックスだけとは限りません。Microsoft PowerPoint の大半の図形は、クリックしてテキストを入力できます。これは、ほとんどの図形に **text frame** が含まれているためです (含まれていない図形もあります)。プログラムで図形内のテキストを操作するには、テキスト フレームから開始します。簡単なマクロを次に示します。

```
Sub AddSomeText()
        ' work with the currently selected shape
        With ActiveWindow.Selection.ShapeRange
                ' the shape's "text container" is the TextFrame
                With .TextFrame
                        ' unless you specify otherwise, .TextRange refers to
                        ' all of the text in the text frame
                        With .TextRange
                                ' add some text
                                .Text = "You can add text to shapes"
                        End With
                End With
        End With

        ' or somewhat more compactly
        With ActiveWindow.Selection.ShapeRange
                TextFrame.TextRange.Text = "This works too"
        End With

End Sub
```

ここで紹介するサンプル マクロのほとんどは、**ActiveWindow.Selection.ShapeRange** から始まっていますが、AppleScript ではこれを再現できません。この状況には、頻繁に直面することが考えられます。Microsoft Office 2004 では、必要に応じてスクリプトの最後に GetSelectedShape() ハンドラをペーストすることができます。このトピックでは記述していないこともありますが、このハンドラはほぼ必ず呼び出されるものであり、常に必要なものです。

```
tell application "Microsoft PowerPoint"
        --get the selected shape via the handler
        set selectedShape to my GetSelectedShape() --paste it in below!

        tell selectedShape
                tell its text frame --needs 'its'!
                        tell its text range --needs 'its'
                                --unless noted, refers to all
                                --the text in the text frame
                                set content to "You can add text to shapes."
                        end tell
                end tell
        end tell

        --or, more compactly
        set content of text range of text frame of selectedShape to ¬
                "This works too!"

end tell
```

最初のテキストの変更を確認する場合は、初回実行時に最後の行を削除してください。最初のテキストはすばやく次のテキストに置換されるので、特定は困難です。`tell` ブロックは Visual Basic for Applications (VBA) の `With` ブロックに相当しますが、これをネストして使用すると、中間のターゲットに対してコマンドを追加する場合に便利です。他の方法として、このような単一のコマンドの場合は、`of` を使用してより短いコマンドにすることもできます。`tell` と `of` は必要に応じて組み合わせることができ、どちらかに統一する必要はありません。

この例には **text frame** と **text range** の 2 つのプロパティが含まれているので、覚えておくと便利です。これらはどちらも、それぞれが含むオブジェクトのプロパティにもクラスにも使用できる用語 (キーワード) です。したがって、`tell` ブロックと `whose` (`where its`) 句には必ず `its` を使用する必要があります。この例では 2 回使用されていますが、`text frame` に対する最初の `its` を省略した場合、スクリプトは `text range` に対してもコンパイルされなくなります。

図形でテキスト フレームがサポートされるようにする

線、矢印、コネクタなど一部の図形には、テキスト フレームを含めることができません。また、**shape** にテキスト フレームが含まれている場合でも、テキスト フレームまたはそのテキストへのアクセスの際にマクロによってエラーがスローされることがあります。この問題は、条件分岐とエラー トラップによって対処します。

```
Sub TextCaveats()

        With ActiveWindow.Selection.ShapeRange

                On Error Resume Next

                ' does it have a text frame?
                If .HasTextFrame Then
                        ' does the text frame have any text?
                        ' see if there's any there
                        ' before trying to alter it
                        If .TextFrame.HasText Then
                           .TextFrame.TextRange.Text = "And here you are, safely"
                        End If
                End If
        End With

End Sub
```

AppleScript では次のようになります。

```
tell application "Microsoft PowerPoint"
        --get the selected shape via the handler
        set selectedShape to my GetSelectedShape()
                --paste it in below!
        try --to avoid error messages if shape
                --has no text frame or no text
                tell selectedShape
                       if has text frame then --does it have one?
                                  --if so, does the text frame
                                  --have any text?
                                  if has text of its text frame then
                                           --needs 'its'
                                           set content of text range ¬
                                                 of its text frame to ¬
                                                 "And here you are safely."
                                  end if
                       end if
                end tell
        end try
end tell
```

実際の運用スクリプトでは、GetSelectedShape ハンドラの後のすべての行を IsSafeToTouch(selectedShape) ハンドラ内に配置し、selectedShape に渡して、ブール値 *true* を返します (条件が異なる場合や on error では *false* を返します)。この処理が終わった後、テキストをメイン スクリプトで設定します。このハンドラをテキストの設定前に呼び出すことができる場所やコンテキストは数多くあります。

```
tell application "Microsoft PowerPoint"
        --get the selected shape via the handler
        set selectedShape to my GetSelectedShape() --paste it in below!
        set check to my IsSafeToTouch(selectedShape)
        if check then
                set content of text range of selectedShape's text frame to ¬
                        "Different text depending on context."
        end if
end tell

on IsSafeToTouch(selectedShape)
        tell application "Microsoft PowerPoint"
                try --to avoid error messages if shape has no
                        --text frame or no text
                        tell selectedShape
                                if has text frame then
                                        --does it have one?
                                        --if so, does the text frame
                                        --have any text?
                                        if has text of its text frame then
                                                --needs 'its'
                                                return true
                                        else
                                                return false
                                        end if
                                else
                                        return false
                                end if
                        end tell
                on error
                        return false
                end try
        end tell
end IsSafeToTouch
```

テキストの書式を設定して文字を選択する

図形のテキストの書式を設定する

テキストの書式設定は簡単です。AppleScript のプロパティと Visual Basic For Applications (VBA) のプロパティは名前も機能も対応しています。

```
Sub TextFormatting()

        With ActiveWindow.Selection.ShapeRange
                ' work with all the text in the shape
                With .TextFrame.TextRange
                        ' Change font formatting
                        With .Font
                                .Name = "Arial"
                                .Size = 24

                                ' make it red
                                .Color.RGB = RGB(255, 0, 0)

                                .Bold = True
                                ' and so on
                        End With
                End With
        End With

End Sub
```

AppleScript では、次のようになります。

```
tell application "Microsoft PowerPoint"
        --get the selected shape via the handler
        set selectedShape to my GetSelectedShape() -- paste it in!
        tell text range of text frame of selectedShape
                tell its font
                        set {font name, font size, font color, bold} to ¬
                                {"Arial", 24, {255, 0, 0}, true}
                end tell
        end tell
end tell
```

AppleScript では、**font** のプロパティを必要な数だけ (ここでは 4 つ)、リストとしてすべて 1 行で設定できます。

メモ これは、対象のオブジェクトに対する `tell` ブロック内でのみ有効で、`of` 形式は使用できません。ただし、プロパティのリストの取得は `of font` を使用していつでも実行できます。

図形の文字を選択する

UI からこのマクロを実行してテストするには、 [ツール] メニューの [マクロ] をクリックします。

このマクロは次の 4 つの部分に分けられます。最初の部分ですべての **character** について新しい **MsgBox** を取得するので、あまり長いテキストは使用しないでください。

- 最初の部分では、MsgBox の各文字を点滅させた後、1 文字おきに青に変えます。
- 2 つ目の部分では、それぞれの **word** を点滅させた後、異なる緑の網かけに変えます。
- 3 つ目の部分では、同じ書式が設定されたテキストのまとまりごとに、テキストと **RGB** 値を表す **Long** を表示します。
- 4 つ目の部分では、テキストの任意の箇所を表示します。ここでは 3 番目の文字の後ろに続く 6 文字を表示します。

```
Sub WorkWithPartOfText()

        Dim x As Long, g As Long

        With ActiveWindow.Selection.ShapeRange.TextFrame.TextRange
                ' access one character at a time
                For x = 1 To .Characters.Count
                        MsgBox .Characters(x)

                        ' make every second character blue
                        ' if x divided by 2 = x mod 2 ...
                        If x / 2 = x \ 2 Then
                                .Characters(x).Font.Color.RGB = RGB(0, 0, 255)
                        End If
                Next

                ' or a word at a time:
                g = 0
                For x = 1 To .Words.Count
                        MsgBox .Words(x)
                                'make every word a different shade
                                g = g + 100
                                If g > 255 Then g = 0
                        .Words(x).Font.Color.RGB = RGB(0, g, 100)
                Next

                ' or a "run" at a time
                ' every change in formatting starts a new run
                ' if the number of runs in a textrange is 1,
                ' you know that all of the text
                ' in the range is formatted identically.
                For x = 1 To .Runs.Count
                        ' For each run, display the text and the Long that
                        ' represents the text's RGB color
                        MsgBox (.Runs(x).Text & " - " & .Runs(x).Font.Color.RGB)
                Next
```

```
                ' or arbitrary selections of text
                ' 6 characters starting at position 3, for example
                MsgBox .Characters(3, 6)

        End With
```

AppleScript では、次に示すように多少変更が必要となります。

```
tell application "Microsoft PowerPoint"
        --get the selected shape via the handler
        set selectedShape to my GetSelectedShape() -- paste it in below!

        tell text range of text frame of selectedShape
                repeat with i from 1 to count (characters)
                        display dialog (get content of character i)
                        --make every second character blue
                        if i / 2 = i div 2 then
                                set font color of font of character i ¬
                                        to {0, 0, 255}
                        end if
                end repeat

                set g to 0
                repeat with i from 1 to count words
                        display dialog (get content of word i)
                        --make every word a different shade
                        set g to g + 100
                        if g > 255 then set g to 0
                        set font color of font of word i to {0, g, 100}
                end repeat

                set AppleScript's text item delimiters to {", "}
                repeat with i from 1 to (count text flows)
                        set theRun to text flow i
                        display dialog (get content of theRun) & " - {" & ¬
                                font color of font of theRun & "}"
                end repeat

                set AppleScript's text item delimiters to {""}
                display dialog (text 3 thru 8 of (get content))

        end tell
end tell
```

`repeat` ループ内のカウンタが `x` から `i` に変更されている点に注意が必要です。文字 `x` と `y` は、**animation behavior** に使用される **property effect** クラスの列挙プロパティの定数として、Microsoft PowerPoint Dictionary に定義されています。したがって、これらの文字は変数としてコンパイルされず、`Expected variable name or property but found application constant or consideration` という AppleScript の構文エラーが返されます。

また、**display dialog** コマンドが **text frame** の内容をターゲットとしている "複合的な" 行では必ず、**content** プロパティの前に **get** を明示的に指定する必要があります。このようにしないと、`Can't make content of text frame... into a string` というエラーが返されます。

メモ このエラーの原因は、PowerPoint で Unicode テキストを使用したことではありません。Mac OS X バージョン 10.3 (Panther) 以降の **display dialog** コマンドでは、Unicode テキストの使用が認められています。`as string` による強制型変換が有効であることから、Unicode テキストがエラーの原因であるように見えますが、この強制型変換が有効なのは、これによって **get** コマンドが実行するような評価が強制的に行われるためです。

VBA の MsgBox と異なり、AppleScript の **display dialog** コマンドでは、**character**、**word**、**text flow** などのオブジェクトの **content** プロパティの取得が必要です。**display dialog** コマンドで、オブジェクト参照自体を取得して使用しようとすると、エラーが発生します。また、この例では既定のプロパティを持つ VBA オブジェクトが使用されています。ここでは、`Words(Index)` などは、既定の **Text** プロパティを持つ **TextRange** を返します。Applescript には既定のプロパティがないため、**get** コマンドを明示的に使用して、`word i` などの内容を取得する必要があります。

VBA では、既定のプロパティと強制型変換を使用することによって、Character、Word、および **Run** を明示的に文字列にキャストしなくても、MsgBox (および **Debug.Print**) を使用することができます。一方、MsgBox も **display dialog** コマンドも (display dialog については Mac OS X バージョン 10.4 (Tiger) 以降) 数値に対応し、それらを文字列に強制型変換します。

最後に、AppleScript ではフォントなどの色を取得しても、VBA のように単一の数値 (Long) として返されるのではなく、3 つの整数値のリストとして返されます。しかし、display dialog ではリストを表示できません。リスト `{0, 200, 100}` の前に文字列を配置するため、`as string` を使用しなくても、そのリストが連結演算子 `&` により、暗黙的に文字列に強制型変換されます。

文字列のリストが明示的または暗黙的に文字列に強制型変換される際、リストの項目の間にはテキスト区切り文字が挿入されます。これは AppleScript のテキスト項目区切り文字であり、スクリプト共通のものです。既定の区切り文字は `""` で、リスト項目の間に何も挿入されません。区切りとして認識できる文字に変更しない場合、{0, 200, 100} のような 3 つの整数値のリストは強制型変換され、`0200100` のように表示されます。区切り文字として `", "` を使用すると `0, 200, 100` となります。さらにその文字列の両側に中かっこ `{", "}` を追加すると `{0, 100, 200}` となり、ちょうど **display dialog** コマンドで取得するリストと同じになります。

テキスト項目区切り文字を変更した場合は、使用後に変更前の文字または既定の `{""}` に戻す必要があります。これは、変更した文字が、Macintosh スクリプト エディタ でスクリプトを実行するアプリケーション内でも、他のすべてのスクリプトに対しても、またすべての [スクリプト] メニューでも、もう一度変更しない限り引き続き使用されるためです。これらは理論上は区切り文字のリストであるため、リストを囲む中かっこ (`{""}`、`{", "}` など) を指定していますが、実際に使用されるのは最初のリスト項目だけであり、中かっこを省略することもできます。

スライド マスタとドキュメント プロパティ

マスタ

マスタは、**slide** オブジェクトの 1 つです。たとえば、現在選択しているスライドのマスタに、新しい四角形の **shape** を追加する場合、Visual Basic for Applications (VBA) では次のように記述します。

```
Sub AddShapeOnMaster()

        With ActiveWindow.Selection.SlideRange
        ' Make a new shape in the slide's Master instead of the slide itself
                Call .Master.Shapes.AddShape(msoShapeRectangle, 10, 20, 30, 40)
        End With

End Sub
```

AppleScript では次のように記述します。

```
tell application "Microsoft PowerPoint"
        --get the selected slide make sure slide pane is active
        set theIndex to slide index of slide of view ¬
                of active window
        set selectedSlide to slide theIndex of active presentation

        --no 'master' property of slide, get slide master of presentation
        set theMaster to slide master of active presentation
        make new shape at theMaster with properties {auto shape type:¬
                autoshape rectangle, ¬
                left position:10, top:20, width:30, height:40}

end tell
```

標準表示の場合、スクリプトの実行前にスライド ウインドウ内の任意の場所をクリックしてアクティブにする (フォーカスを設定する) 必要があります。このようにしないと、`slide of view` が存在しないために次の行でエラーが発生します。選択以外の方法でスライドを指定すれば、この問題は起こりません。

Dictionary には、VBA の .**Master** に相当する **slide** クラスの `master` プロパティがありません。しかし、**presentation** 内のスライドのマスタは、プレゼンテーションの **slide master** プロパティと同じであり、これを使用できます。選択したスライド (これに対するプロパティはありません) のプレゼンテーションは **active presentation** である必要があります。そうでない場合は選択できません。

Microsoft Visual Basic Editor ヘルプの **Shape** クラスに関する項目では、**AddShapes** メソッドで **AutoShape** が追加されることが確認できます。ここでは特定の定数 **MsoAutoShapeType** と、位置とサイズを示す 4 つの整数 (*Left*、*Top*、*Width*、および *Height*) が指定されています。これは上の VBA コードで指定している内容と同じです。

Dictionary (Drawing Suite) の **shape** クラスのエントリには、**auto shape type** プロパティ (読み取り/書き込み) があります。このプロパティを見ると、定数 (long 型) が列挙されています。これは、Visual Basic Editor ヘルプとオブジェクト ブラウザで確認できる **MsoAutoShapeType** の列挙定数とよく似ています。さらに、そこには定数 *autoshape rectangle* が含まれています。一方、**shape** クラスの独立した **shape type** 列挙プロパティには四角形に相当または関連するものがありません。したがって、**auto shape type** 列挙、特に定数 *autoshape rectangle* が必要です。

make new shape に必要な他のプロパティは、**left position**、**top**、**width**、および **height** です。スクリプトは正しく機能し、新しいスライドを挿入すると、同じ四角形が現れます。これで、スライド マスタへの図形の追加に成功したことがわかります。

Microsoft PowerPoint 2004 for Mac の UI では、"デザイン" と呼ばれる複数のマスタを所有できます。デザイン コレクションは、過去 3 バージョンの Microsoft Windows 向け PowerPoint VBA に実装されています。ただし、Macintosh 向け PowerPoint VBA には提供されていません。AppleScript モデルは Macintosh VBA モデルの一部であるため、デザイン コレクションは存在しません。

そこで、VBA の **ActivePresentation.ApplyTemplate (FileName)** メソッドに相当する **apply template** コマンドを使用して、テンプレートからデザインをプレゼンテーションに適用します。テンプレート (**file name** パラメータで表されます) に複数のデザインが含まれる場合は、それらが適用される可能性があります。現在所有するデザインを後から確認することはできません。

ドキュメント プロパティ

PowerPoint および他の Microsoft Office アプリケーションには、便利な情報を含む一連のドキュメント プロパティがあります。これらのプロパティにアクセスするには、必要なプロパティの名前を把握する必要があります。次のマクロでその一覧を取得できます。

```
Sub ListBuiltInProperties()
        Dim x As Long
        On Error Resume Next
        Debug.Print "BEGIN ==================================="
        With ActivePresentation.BuiltInDocumentProperties
                 For x = 1 To .Count
                          Debug.Print "Property Number: " & CStr(x)
                          Debug.Print "Property Name: " & .Item(x).Name
                          Debug.Print "Property Value: " & .Item(x).Value
                 Next
        Debug.Print "END ====================================="
        End With
End Sub
```

AppleScript では次のようになります。

```
tell application "Microsoft PowerPoint"
        set theLog to "BEGIN ============================" & return
        set allDocProps to every document property of active presentation
        repeat with i from 1 to count allDocProps
                set theDocProp to item i of allDocProps
                set theLog to theLog & ¬
                        "Property Number: " & i & return
                set theLog to theLog & ¬
                        "Property Name: " & name of theDocProp & return
                set theLog to theLog & "Property Value: " ¬
                        & value of theDocProp & return
        end repeat
        set theLog to theLog & "END ========================" & return
end tell
theLog
```

AppleScript クラスの名前に "ビルトイン" はありませんが、**document property** はビルトインのものだけを参照します。AppleScript では、ドキュメント プロパティは **application** の要素であるため、どのドキュメント プロパティもビルトインのクラスをすべて取得します (30 個あります)。このクラスは、Microsoft Office Suite のどの Office アプリケーションにも存在します。UI でカスタム プロパティを追加しても、すべてのドキュメント プロパティに追加されるわけではありません。

必要なプロパティの名前がわかったら、それを使用できます。VBA で、**Template** のような特定の **DocumentProperty** の値を表示するには次のようにします。

```
With ActivePresentation
        MsgBox .BuiltInDocumentProperties("Template").Value
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft PowerPoint"
        set templateValue to get value of document property ¬
                "Template" of active presentation
        if templateValue  missing value then
                display dialog templateValue
        else
                display dialog "There is no Template: ¬
                        this presentation is " & ¬
                        "based on a Blank New Presentation" with icon 2
        end if
end tell
```

自分で作成またはプロジェクト ギャラリーからインポートしたテンプレート以外のものは、すべて `template` プロパティの値が `missing value` (存在しない) になります。これは VBA にも当てはまり、空の **MsgBox** が返されます。AppleScript ではエラー メッセージが返されるため、エラーをトラップして適切な メッセージ ダイアログ ボックスを表示することをお勧めします。

Template プロパティは読み取り専用であり、変更できません。**Author** のようなプロパティは、読み取り/書き込みが可能です。VBA の読み取り/書き込みプロパティの値を変更するには、次のようにします。

```
With ActivePresentation
        .BuiltInDocumentProperties("Author") = "Your Name Here"
End With
```

ここにはユーザーの名前が既定で設定されています (すべての Office アプリケーションは、Microsoft Entourage アドレス帳の [自分の連絡先] を参照し、そこにある名前を **Author** ドキュメント プロパティに使用します)。ただし、ビジネス上の目的などで別の名前に変更する場合は、作成した商用プレゼンテーションに対して次の AppleScript を実行できます。

```
tell application "Microsoft PowerPoint"
        set value of document property "Author" of active presentation to ¬
                "Phyllis Harris, Manager"
end tell
```

確認するには、 [ファイル] メニューの [プロパティ] をクリックし、 [ファイルの概要] タブをクリックします。

PowerPoint AppleScript のまとめ

随時 Visual Basic for Applications (VBA) のオブジェクト、プロパティ、メソッドと、AppleScript Dictionary のクラス、プロパティ、要素、コマンドを常に比較し、相当するものを確認してください。

PowerPoint Dictionary には、必要なクラスとプロパティが不足している場合が多いため、Microsoft PowerPoint での作業は Word や Excel で作業する場合に比べて難しくなりますが、クリエイティブに考えることで別の選択肢を見つけることができます。