



Flash MX 2004 actionscript

Derek franklin/job maker

Training from the source

本ドキュメントは、macromedia PRESS「FLASH MX 2004 actionscript / training from the source」(derek franklin / jobe markar)のレッスン1 INTRODUCTION ACTIONSCRIPTをヒム・カンパニーの永井勝則が独自に訳したものです。

なお、レッスン1 INTRODUCTION ACTIONSCRIPTは、

http://www.macromedia.com/devnet/mx/flash/articles/astfts_excerpt.html

で astfts_lesson1.pdf として公開されています。

またソースファイルも、上記ページから astfts_samples.zip としてダウンロードできます。

「FLASH MX 2004 actionscript / training from the source」は

http://www.amazon.com/exec/obidos/ASIN/0321213432/qid=1096769063/sr=2-1/ref=pd_ka_2_1/104-5378218-6466362

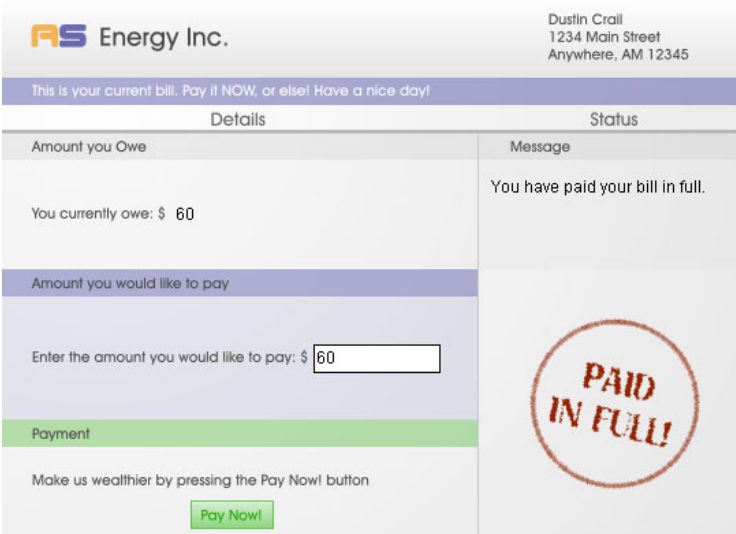
で購入できます。

Introducing ActionScript


Lesson 1

偉大な関係の始まりをご紹介します。準備はいいですか、新しいベストフレンド、ActionScript です。我々は、皆さんが特にその関係を深く追い求めれば求めるほど、ActionScript は満足のいく友人であると実感されることを確信します。たとえ皆さんが独創的なスクリプティングを試そうと思っていないとしても、実際に役立つ ActionScript の知識は、あらゆるアイデアを光り輝かせます。アイデアによって、皆さんは数多くのレベルで皆さんのユーザーと双方向的にやりとりのできるダイナミック・コンテンツを作成できます。何よりも、皆さんのアイデアがプロジェクトの目的をかなえ、役立つモデルに成長するのを見て、満足できるのです。

高度にインタラクティブな電子支払いシステムを立案、作成しテストします。



The screenshot shows a web interface for AS Energy Inc. The header includes the company logo and name, and contact information for Dustin Crail at 1234 Main Street, Anywhere, AM 12345. A purple banner reads "This is your current bill. Pay it NOW, or else! Have a nice day!". Below this is a table with two columns: "Details" and "Status".

Details	Status
Amount you Owe	Message
You currently owe: \$ 60	You have paid your bill in full.
Amount you would like to pay	
Enter the amount you would like to pay: \$ <input type="text" value="60"/>	
Payment	
Make us wealthier by pressing the Pay Now! button	
<input type="button" value="Pay Now!"/>	

このレッスンでは、これまで ActionScript を使用されていた方にそのバージョン 2.0 をご紹介いたします。新規ユーザーには、ActionScript を学ぶ理由を、その内部の仕組みとともにご覧いただきます。少し気が進まないときは、そこに飛び込むことが最良の場合もあります。そことは、レッスンが終わるまでにインタラクティブなプロジェクトを作成しテストする、まさにこの場での作業のことです。

学習すること

このレッスンでは、以下の作業をおこないます。

- ・ ActionScript.2.0 について学び、ActionScript1.0 とどのように異なるか学習します。
- ・ ActionScript を学ぶ利点を明らかにします。
- ・ ActionScript エディターを操作し、使用方法を学びます。
- ・ スクリプト・エレメントについて学びます
- ・ プロジェクトを計画します。
- ・ 最初のスクリプトを書きます。
- ・ スクリプトをテストします。
- ・ スクリプトをデバッグします。

かかる時間

このレッスンには終了までおよそ 1 時間半かかります。

レッスン・ファイル

Lesson01/Assets/electricbill fla

完成プロジェクト

electricbill2 fla

electricbill3 fla

ActionScript のバージョン 2.0 への成長

ある期間 Flash に関わって来ている皆さんは、Flash が、アニメーションする Web グラフィックスやインタラクティブなボタンを作成するのに使われるシンプルなマルチメディア・ツールから、外部 MP3 ファイルを再生できたり、グラフィックスのロード、ビデオ再生やデータベースとのやり取りなどが行える最強のマルチメディア・ツールに発展してきたことをご存知でしょう。

世界中の Flash 開発者の情熱的で革新的なフォーラムは、この発展を大いに推し進めています。Flash の開発を常に新しい高みへと押し上げることで、彼らは我々に、その可能性だけでなく、できることの限界も知らしめています。

幸いなことに、Flash の新バージョンごとに、Macromedia 社は、その限界に取り組み懸命に努力を重ね、さらにクールで、簡単で、効果的な手法を可能にさせるツールを開発者に提供しています。

Flash 5 の登場以来、ActionScript は Flash ベースのコンテンツが新たなる高みに上ることを可能にしましたが、それとともに障害や限界が分かってきました。

Flash 5 player で ActionScript を実行させると、スピードが遅くなる傾向が見られました。他のスクリプティング/プログラミング言語ではミリ秒だけで足りるタスクも、ActionScript では何秒かかかっていた。ゲーム・プログラマーにたずねると、彼らは、その処理速度は一生みたいに思えると答えるでしょう。そして実際、使用できるインタラクティブ性にも限界がありました。

遅い処理速度に加え、Flash 5 の ActionScript はパワフルであった一方、融通性に欠けました。さらに複雑で管理しやすい Flash アプリケーションを作成できるオブジェクト指向プログラミング技術は、たやすい方法では実現できませんでした。

Flash MX の ActionScript は、正式には ActionScript1.0 を拡張した 2.0 ではないですが、ActionScript1.5 と呼んでもさしつかえないもので、Flash 5 ActionScript の抱える処理速度の問題に取り組んでいます。加えて、Flash MX の ActionScript は、カスタム・オブジェクト・クラスや継承の作成など、一般的なオブジェクト指向プログラミングの技術を使用できるように拡張されました。これは正しい方向への明らかに大きな一歩ではありましたが、依然として改善の余地は残されていました。

Flash MX 2004 で Macromedia 社は ActionScript2.0 を登場させました。ActionScript2.0 は新

しい可能性とともに、さらに重要な、新しいシンタックス、コードの新しい構造、作用を伴っています。

次のセクションで論じるように、少ないものの、その変化は `ActionScript` をプロフェッショナル・グレードのプログラミング言語の領域へ入り込ませるものです。そして近年 `Flash` アプリケーションに人々が要求するものを考えると、この変化は明らかに正しい方向を示しています。

NOTE `ActionScript` を学び始めたばかりならば、このセクションは（おそらく意味がありませんので）飛ばして、次へお進みください。

ActionScript1.0 と 2.0 の相違点

`ActionScript1.0` を使い慣れていらっしゃるなら、`ActionScript2.0` はそれによく似ていると思われるでしょう。しかし小さいですが重要な違いです。このセクションでは、その違いについて実験し、身に着けるのが大変だった `ActionScript1` の知識を学び、それを `ActionScript2.0` に置き換えます。

これらの違いについて見ていく前に、ひとつ言わせていただきます。もし皆さんが今ご存知の方法(`ActionScript1.0` のプログラミング)を続けていきたいと思われるのであれば、どうぞそうなさってください。`ActionScript1.0` のシンタックスは `Flash MX 2004` でも動作します。しかし、皆さんがされた選択ではありますが、我々はいくつかの理由から `ActionScript2.0` を学ぶ時間を取られることをおすすめします。

まず、`Flash` のあるバージョンがリリースされると、`ActionScript1.0` はその後サポートされなくなるという時が来るかも知れません。今バージョン 2.0 を学ぶのに費やす時間は、将来何らかの方法で費やさねばならない時間になるでしょう。そして保証はできませんが、`ActionScript` がいつの日かバージョン 3.0 にジャンプするとは思えません。それは、現行のバージョン 2.0 がテストに時間をかけたプロフェッショナルなプログラミング・コンセプトにより構築されたものだからです。実際問題として、シンタックスの相違は別にして、`ActionScript2.0` のコードを書くことは、`Java` コードを書くことと大した違いはないのです。

その通りです。皆さんが `ActionScript2.0` を学ぶ時間を取るのなら、その知識を、非常に強力なクロス・プラットフォームのアプリケーションが標準である、`Java` 世界にすぐにでも移すことができるのです。まさに一石二鳥です。

2 つめに、その本質的な要求により、`ActionScript2.0` は、さらに効果的で組織化された、

良いコードになります。まもなくご覧いただくように、ActionScript2.0 は、何かするのに厳格な要求を必要とします。ActionScript1.0 のようには許されません。これは良くないことのようにですが、実際には、バグの発見を苦痛にさせ、何ヶ月か後のプロジェクトの更新作業を骨の折れるものにする、不注意によるスクリプティングのミスを避けることができます。

最後に、速度が重要な場合、ActionScript2.0 は、1.0 の3倍から7倍表示が速くなるのはありがたいことです。このスピードの増加は、さらにパワフルな Flash アプリケーションの開発を促進することでしょう。

では、次に ActionScript1.0 と 2.0 に見られる主な違いを示します。

大文字と小文字の区別

ActionScript1.0 では、以下の変数名は同じ変数を参照しました。

myVariable

MyVariable

しかし、ActionScript2.0 では、大文字小文字の違いにより2つの別の変数として扱われます。1つめの変数は小文字で始まり、2つめは大文字で始まっています。実際 ActionScript2.0 では次のものは全て異なる要素として扱われます。

myName

MyName

MYNAME

myname

myNAME

大文字と小文字の区別というこの決まりは、ActionScript2.0 では、インスタンス名、キーワード、メソッド名など、全ての要素に適用されます。よって、このシンタックスは再生中の全てのサウンドを停止させますが、

```
stopAllSounds();
```

このシンタックスはエラーの原因になります。

```
stopallsounds();
```

TIP 大文字小文字のエラーをテストする簡単な方法は、アクションパネルのシンタックスチェックボタンを押すことです。大文字小文字のミスマッチによるエラーは出力ウィンドウに表示されます。

厳格な型づけ

変数はデータを保持するために使われます。このデータには、テキストのストリング、数

値、真偽値、ムービークリップ・インスタンスなどのオブジェクトへの参照など、多くの形式があります。ActionScript1.0 では、変数を作ると Flash は自動的にそれに型を割り当てていました。この例では、Flash は右辺の値は数値型であると理解しました。

```
myVariable = 36;
```

Flash は今でもこの変数は数値型であると認識しますが、ActionScript2.0 では、変数の型設定をさらに管理することができる、**厳格な型づけ**が導入されています。

ActionScript2.0 で変数を作る時は、変数を作るだけでなく同時に型を割り当ててこのシンタックスを使用します。

```
var myVariable:Number = 36;
```

ご覧のように、このシンタックスは ActionScript1.0 で使い慣れていたものとそんなに違いがありません。違いは var キーワードの追加と、この変数は数値を保持する、そのことを:Number シンタックスで表すという、明示的な宣言です。

文字列を保持する変数はこのようになります。

```
var myOtherVariable:String = "Hekko";
```

型を通常の変数に割り当てることに加え、厳格な型づけでは、オブジェクト・インスタンスを作成するときにも使用されます。

```
var myArray:Array = new Array();
```

そしてファンクション定義にも使用されます。

```
function myFunction (name:String, age:Number)
```

NOTE 両方の型の厳格な型づけについては、本書の後半でさらに深く取り扱います。

このように余計にキーを打つことで何が強化されるのでしょうか？ それにはいくつかあります。

コードヒント(手短かに論じます)を表示させるアクションパネルの機能に慣れている方なら

ば、厳格な型づけによって、名前をつけた要素にコードヒントが表示されるようになるとありがたいと思われるでしょう。例えば、以下のシンタックスを使って Array オブジェクトを作ったとすると、

```
var myArray:Array = new Array();
```

アクションパネルはその場所から myArray への参照は Array オブジェクトへの参照であると認識し、そのオブジェクトを記述するときはいつでも自動的に Array オブジェクトのコードヒントを表示します。この新機能により、Flash MX でオブジェクトのコードヒント表示を可能にしていた接尾辞(_array,_xml,_color など)の必要性がなくなる場合もあります。新シンタックスでは少し多くのキーストロークが必要ですが、一方で要素名に接尾辞をつける必要はなくなるのですから、その分差し引きマイナスになります。

NOTE 新機能により接尾辞は必要なくなる場合もあると述べました。視認要素(ムービークリップ・インスタンス、ボタン、テキスト・フィールド、コンポーネントなど)は、データ要素と同じようには作成されず、名前も付けられません。この理由により、これらの要素の名前(_mc,_btn,_txt)の接尾辞は依然有用です。本書では、視認要素に限り接尾辞を使用しています。

コードヒント表示だけが厳格な型づけの利点ではありません。変数が保持する型を示すことによって、Flash player が表示するまでかかる時間を節約できるのです。Flash player がアプリケーションを起動させると、Flash はそれぞれの変数が所有する型の情報をずっと追う必要があります。厳格な型づけを使用したなら、この過程は単純なものになりますし、それは Flash player への処理負担を減らすことにもなります。結果として、スクリプトは格段に速く実行されることとなります。アプリケーションの速度を上げたいのならば、厳格な片付けを使いましょう。

最後に、厳格な型づけは、コード内にあるバグの発見を助ける大変単純な方法となります。以下は例です。

```
var favoriteBand : String;
```

このスクリプト行はストリングを保持する厳格に型づけされた favoriteBand という名前の変数を作成します。

後からこの変数に値を与えたり、シンタックスを使って新しい値を与えます。

```
favoriteBand = "The Beatles";
```

しかし、スクリプトのどこかでこのシンタックスのようなスクリプトを書くと、

```
favoriteBand = 46;
```

ムービーを書き出し（コンパイル）しようとする、出力ウィンドウが開いて、エラーが表示されます。これは変数に正しくない型の値を代入しようとしたことを表示するエラーです。上の例では、ストリング値を保持するよう設定された変数に、数値（Number 型）を代入しようとした。

この機能は、変数に値を代入するとき、犯すかもしれない単純なミスから起こるバグを避ける上で非常に役に立ちます。

クラス構造

ActionScript2.0 での最大の変化はおそらくオブジェクト指向プログラミングが実行される方法です。例えば、ActionScript1.0 では、オブジェクトのカスタム・クラスは以下のように定義しました。

```
_global.Person = function (name, age){  
    this.name = name;  
    this.age = age;  
}
```

ActionScript2.0 では同じクラスは以下のシンタックスを使って作成されます。

```
Class Person {  
    var name : String;  
    var age : Number;  
    function Person (name:String, age:Number){  
        this.name = name;  
        this.age = age;  
    }  
}
```

新しい class キーワードが使われていることに注目してください。このシンタックスは、Java でオブジェクトのクラスを作成するシンタックスに非常に似ています。このシンタックスについてさらに学ぶようになると（この点は詳しく論じませんが）、ActionScript1.0 を超える利点がすぐに理解できるようになります。

ActionScript1.0 と 2.0 で、オブジェクトのカスタム・クラスを作成する際のもうひとつの小さな違いは、クラス定義を含むスクリプトは、それ自身の外部.as ファイルとして存在しなければならないということです。このことは、Person クラスを定義するスクリプトは、Person.as という名前でも外部.as ファイルとして保存される必要があるという意味です(as フ

ファイルは、.as という拡張子のついたテキストファイル以外の何者でもありません)。これが Actionscript1.0 との違いで、1.0 では、ムービークリップのタイムラインのフレーム 1 にクラス定義のコードを配置できました。この方法は、Flash7 player のコンテンツを作成する際にはもう使用できません。クラス定義は外部.as ファイルに存在しなくてはならないのです。Lesson7 の”カスタムクラスの作成”でこの機能について論じます。今はその違いを覚えておいてください。

NOTE フレームやボタンなどには今でもスクリプトを配置できますが、クラス定義はそれ自身の.as ファイルに存在しなくてはなりません。

できるだけ簡単に.as ファイルを作成するために、Flash MX 2004 では、アクションパネルから分離し、.as ファイルを作成したり保存したりするためだけに使用される、単体の ActionScript エディターが提供されます。このレッスンの終わりにアクションパネルと単体の ActionScript エディターをみてください。

ActionScript1.0 と ActionScript2.0 には他にも相違点がありますが、ActionScript1.0 で作業されてきた方には、この短い概説で 2.0 を始める十分な説明になると思います。

ActionScript1.0 と 2.0 の類似点

ここまで、ActionScript の 2 つのバージョンについて相違点ばかり語ってきました。しかし同じことはないのか？ 苦労して習得した今の ActionScript の知識は、ActionScript2.0 に適用できるのか？ 幸いなことにたくさんあるのです。

ムービークリップはこのシンタックスで制御できます。

```
myMovieClip_mc.gotoAndPlay(15);
```

条件文も同じように作成できます。

```
If (myNumber1 + myNumber2 == 20 && enabled != true){  
    //actions;  
}
```

またループ・ステートメントはこのようです。

```
for( i=0; i<= myVariable; ++i){
```

```
//actions;  
}
```

正規表現は同じように作成します。

```
MyVariable = (myNumber1/2) + (myNumber2 + 15)
```

そしてスクリプトは、on() や onClipEvent() イベントハンドラを使って、ボタンやムービークリップ・インスタンスに割り当てることができます。

なぜ ActionScript を学ぶのか？

もしあなたが Flash 開発者ならば、ひとつ確かなことがあります。それは、どんなにすばらしいアニメーション・スキルをお持ちであろうとも、今やそれだけでは十分ではないということです。ActionScript をしっかり理解することは必須で、ActionScript なしでは、入門的なインタラクティブティだけになってしまいます。ActionScript を深く知ることで、以下のことができるようになります。

- ・ その人だけのユーザー体験を提供できる。
- ・ ムービークリップやそのプロパティ管理をさらに行えるようになる。
- ・ ムービー内でプログラミングにより要素をアニメーションできる。つまり、タイムラインを使わずにアニメーションできる。
- ・ Flash 内外からデータを取得して、フォームやチャット・プログラムなどが作成できる。
- ・ 時間や現在時刻の経過に応答するダイナミックなプロジェクトが作成できる。
- ・ サウンドの音量やパンをダイナミックにコントロールできる。
- ・ などいろいろ。

これらの利点に加え、Flash コンテンツを通して見たり双方向にやりとりすることは、Web 体験以上のものになる可能性があります。Flash は、ブラウザから独立した自分で起動するアプリケーションやミニプログラムを作ることができます。これは、ゲーム作成に使用したり、アプリケーションを学ぶ人々が増える可能性を意味します。あなたももしそうしたいのならば、少なくとも中級レベルの ActionScript の知識を身につける必要があります。

ActionScript の要素

ActionScript は、あなたの理解と Flash の理解を橋渡しする言語です。Flash プロジェクト内では、ActionScript によって、アクション指向の命令(これをやれ)でも、論理指向の命令(それをやる前に分析せよ)でも行えます。全ての言語のように、ActionScript にも、用語、句点、構造など多くの異なる要素があります。それらは全て、Flash プロジェクトにさせたい動作をするように適切に使用しなければいけません。もし ActionScript を正しく使用しないと、インタラクティブティは起きず、意図した動作はしないでしょう。本書では、これ

ら要素の多くを、論理的ステートメントや正規表現などの要素と同様に、詳細まで扱っていきます。

ActionScript の仕様を理解する手始めとして、典型的なスクリプトを構成する重要な要素を多く含む以下のサンプルをご覧ください。その後で、スクリプトが実行しようとしている要素や役割について論じます。

ここではこのスクリプトはボタンに割り当てられてると仮定します。

```
On(release){
    //マグカップの値段の設定
    var mugCost:Number = 5.00;
    //地域販売税のパーセンテージの設定
    var taxPercent:Number = .06;
    //税の総量の決定
    var totalTax:Number = mugCost * taxPercent;
    //総額の決定
    var totalCost:Number = mugCost + totalTax;
    //カスタム・メッセージの表示
    myTextBox_txt.text = "The Total cost of your transaction is " + totalCost;
    //cashRegister_mc ムービークリップ・インスタンスにフレーム 50 を送る
    cashRegister_mc.gotoAndPlay(50);
}
```

最初はラテン語のように見えるかも知れませんが、要素のいくつかが分かるようになると、すぐに理解できます。

NOTE 他のスクリプト要素(例えば、オブジェクト、ファンクション、ループ、プロパティ、メソッド)は本書で詳細に論じていきます。

イベント

イベントはムービーの再生中に生じ、ある特定のスクリプトの実行を引き起こします。サンプル・スクリプトでは、スクリプトを引き起こすイベントは `on(release)` です。このイベントは、このスクリプトが割り当てられているボタンが離されると、スクリプトが実行されることを示します。あらゆるスクリプトはイベントにより引き起こされ、ムービーは、押されたボタンからテキストフィールド内の変化、再生が終了したサウンドまで、無数のイベントに反応することができます。イベントについては、Lesson2 の “ イベントハンド

ラの使用”で論じます。

アクション

アクションはスクリプトの心臓部を形成します。ひとつのアクションは通常、Flash に、設定せよ、作成せよ、変えろ、ロードしろ、消去しろと命令する行のことをいいます。

以下はサンプル・スクリプトからのアクションの例です。

```
var mugCost:Number = 5.00  
cashRegister_mc.gotoAndPlay(50);
```

1 行めは、mugCost という名前の変数を作成し、型を数値(その変数は数値を保持することを示します)にし、変数の値を 5.00 に設定します。2 行目は、cashRegister_mc ムービー・クリップのインスタンスに、そのタイムラインのフレーム 50 で再生を開始するように伝えます。

一般的に、中括弧で挟まれたスクリプトの行のほとんどは、アクションです。これら行は通常セミコロンで分けられます(句点については手短かに論じます)。

演算子

演算子は多くのシンボル(=,<,>,+,-,*,&&など)を含み、いろいろな方法でスクリプト内の 2 つの要素をつなぐために使われます。以下の例をご覧ください。

- var taxPercent:Number = .06;は.06 という数値を taxPercent という名前の変数に代入しています。
- amountA < amountB は、amountA が amountB よりも小さいかどうかたずねます。
- value1 * 500 は value1 を 500 倍乗じます。

キーワード

ActionScript のシンタックスの中には、特殊な目的のために予約された言葉があります。よってそれらは変数やファンクション、ラベル名として使用はできません。例えば、on という言葉はキーワードで、スクリプトの中で、スクリプトを引き起こすイベントを示すためだけに使用できます。on(press)、on(rollover)、on(rollout)などです。そこに意図された目的以外のためにスクリプト内でキーワードを使おうとすると、エラーが生じます。他のキーワードは、break、case、class、continue、default、delete、do、dynamic、else、extends、finally、for、function、get、if、implements、import、interface、in、instanceof、new、null、private、public、return、set、static、switch、this、try、typeof、undefi

ned,var,void,while,with などがあります。

データ

ダイナミックなスクリプトはほとんど常に、その実行中に、様々なデータを作成し、使い、更新します。変数は、スクリプトに見られる最も一般的なダイナミック・データであり、ユニークな名前を与えられた、データの分身です。変数を作り値を代入すると、スクリプトの中に変数名を挿入するだけでどこからでもその変数にアクセスできます。

NOTE 変数名は大文字小文字の区別があります。myVariable と MyVariable とは同じではありません。

サンプル・スクリプトでは、mugCost という名前の変数を作成し、5.00 という値を代入しました。スクリプトの後方で、その変数名はその変数が含む値を参照するために使用されています。

中括弧

一般的に、開いた中括弧と閉じた中括弧の間に書かれたものは、アクションか、引き起こされたときにスクリプトが必要とするアクションのセットです。中括弧はこんな風に言っています。“この結果として、{こうしろ}”。

```
on(release){  
    //マグカップの値段の設定  
    var mugCost:Number = 5.00;  
    //地域販売税のパーセンテージの設定  
    var taxPercent:Number = .06;  
}
```

セミコロン

スクリプトのほとんどの最後に見られるセミコロンは、ひとつのイベントの結果として実行される必要がある、多数のアクションを分けるために使われます。一文の中で考えを分けるためにセミコロンを使うのとよく似ています。以下の例は6つのアクションを示し、セミコロンで分けられています。

```
var mugCost:Number = 5.00;  
var taxPercent:Number = .06;  
var totalTax:Number = mugCost * taxPercent;  
var totalCost:Number = mugCost + totalTax;  
myTextBox_txt.text = “The Total cost of your transaction is “ + totalCost;
```

```
cashRegister_mc.gotoAndPlay(50);
```

ドット・シンタックス

ドット(.)はスクリプト内で2つの方法で使われます。ひとつは特定のタイムラインへのターゲット・パスを示すためです。例えば、_root.usa.Indiana.bloomington は、メイン (_root) タイムライン上の usa という名前のムービークリップ内にある、indiana という名前のムービークリップ内にある、bloomington という名前のムービークリップを指しています。

ActionScript はオブジェクト指向言語ですので、ほとんどのインタラクティブなタスクは、オブジェクトの特性(プロパティ)を変更したり、オブジェクトに何かするように伝えたり(メソッドの呼び出し)すことでなされます。プロパティを変更したりメソッドを呼び出したりする際、ドットはオブジェクトの名前とプロパティや動作するメソッドとを分けるために使用されます。例えば、ムービークリップはオブジェクトです。wheel_mc という名前のムービークリップ・インスタンスのローテーション・プロパティを設定するためには、以下のシンタックスを使います。

```
wheel_mc._rotation = 90;
```

ドットによってどのようにオブジェクト名とそこに設定されたプロパティが分けられているか注意してみてください。

同じムービークリップに再生を伝える、つまり play()メソッドを呼び出すには、以下のシンタックスを使います。

```
wheel_mc.play();
```

もう一度言いますと、ドットはオブジェクト名と呼び出されるメソッドを分けているのです。

括弧

括弧は ActionScript では様々な方法で使用されます。最も多いのが、アクションが実行時に使用する特定の値を設定する括弧です。サンプル・スクリプトの最後の行をご覧ください。ここでは、cashRegister_mc ムービークリップ・インスタンスにフレーム 50 に移動して再生するように告げています。

```
cashRegister_mc.gotoAndPlay(50);
```

括弧内の値が 50 から 20 に変わると、アクションは同じ基本タスクを実行します(cashRegister_mc ムービークリップ・インスタンスを特定のフレームに移動させます)。そ

れは新しい値に基づいてなされます。括弧は括弧内で特定されたことに基づき動作するようにアクションに伝える方法です。

クォテーション・マーク

クォテーション・マークはスクリプト内のテキスト・データを示すために使用されます。テキストは実際のスクリプト内で使用されるので、クォテーション・マークは、スクリプトが命令（データのかたまり）か実際の言葉かを区別するためだけに使われます。例えば、Derek(“”なし)はデータのかたまりの名前を示します。他方、“Derek”は実際の言葉”Derek”を示します。

コメント

コメントはふたつのスラッシュ(//)を行頭に置かれるスクリプトの行です。スクリプトを実行すると、Flash はコメントを含む行を無視します。コメントは、スクリプトが実行中にその時点で行っていることについての説明書きのことです。コメントによって、スクリプトが書かれた何ヶ月も後になってもスクリプトを見直し、隠された論理のアイデアを得ることができます。

また以下のシンタックスで複数行のコメントを作成できます。

```
/*everything between  
here is considered  
a comment */
```

インデント/空白

絶対に必要という訳ではないですが、コード内にインデントと空白を設けるのは良い考えです。

```
on (release) {  
var mugCost:Number = 5.00  
}
```

は以下と同じことを実行します。

```
on (release) {  
    var mugCost:Number = 5.00  
}
```

しかしコードをインデントすることで、読みやすくなります。中括弧内はインデントして、その部分のコードはコード・ブロック、すなわち、同時に実行される、コードの大きなか

たまりを表すということにしておくのは良いルールです(アクションパネルの自動フォーマットは皆さんに代わってこのことに注意を払っています)。コード・ブロックの中に別のコードをネストすることもできます。この考えは練習を積むにしたがって分かるようになってきます。

ほとんどの場合、スクリプト内の空白は無視されます。例えば、

```
var totalCost:Number = mugCost + totalTax;
```

は以下と同じことを実行します。

```
var totalCost:Number =mugCost+totalTax;
```

プログラマーの中には、空白はコードを読みやすくすると思っている人もいれば、空白を入れたことでスピードが落ちると信じている人もいます。ほとんどの場合、選択は本人がします。ふたつほど例外があります。変数名は空白を含んではいけませんし、オブジェクト名とそれに関するプロパティ、メソッドの間に空白をいれてもいけません。このシンタックスはオーケーです。

```
myObject.propertyName
```

これはだめです。

```
myObject. propertyName
```

加えて、変数を作る時に使われる `var` キーワードと、実際の変数名の間に空白があっては いけません。以下は正しいです。

```
var variableName
```

しかし以下は間違いです。

```
varvariableName
```

アクションパネル/アクションスクリプト・エディターを使う

本書のはじめの方では、スクリプトを書くとき明らかに集中が必要です。Flash で使うことになるツールに馴染むのは良い考えです。初めての練習として、ここでは、アクションスクリプト・エディターでスクリプトを書く基本を学んでいきます。

NOTE 本書の目的は *ActionScript* を学ぶことであって、*Flash* インターフェイスの使い方を学ぶことではありません。この説明は簡単なものに留め、本書を通して皆さんの上達の助けとなる十分な情報を提供することとします。アクションパネルの多くの機能を広く知りたい場合は、*Macromedia Flash Visual QuickStart Guide* をご覧ください。

- 1) **Flash** を起動させ、**ファイル>新規**を選択し、**選択リスト**から **Flash ドキュ**

メントを選択、OK をクリックします。

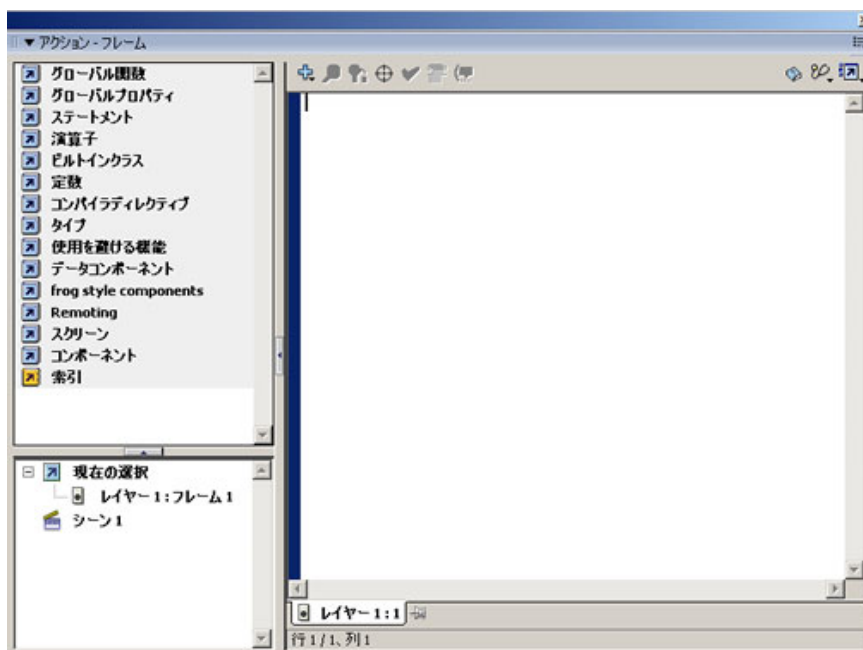
このステップは新しい Flash ドキュメントを作成します。通常新規ドキュメントに名前をつけ保存するのは良い考えです。では次に進みます。

- 2) ファイルメニューから、名前をつけて保存を選択します。現れるダイアログボックスで、このファイルを保存したいハードドライブ上のフォルダを選んで指定します(場所はどこでもかまいません)。MyFirstFile.fla の名前をつけ、保存ボタンを押します。

保存ボタンを押すと、ドキュメント・ウィンドウの上にあるタブに新しいファイル名が表示されます。

- 3) ウィンドウ>開発パネル>アクションを選んで、アクションパネルを開けます。

アクションパネルの様々な部分を見てみましょう。



スクリプト・ペインは ActionScript を加えていく場所です。ワープロのようにこのウィンドウに入力します。このウィンドウに現れるスクリプトは、Flash オーサリング環境で現在選択している要素によって変わります。例えば、フレーム 10 のキーフレームを選択すると、そのフレームにスクリプトがなければそこにスクリプトが書けるようになります。そのフレームにスクリプトが既にある場合は、編集用の表示がされます。

アクション・ツールボックスには ActionScript 要素のカテゴリー別リストがあります。ア

アイコン（矢印のついた本）をダブルクリックすることで、リストのカテゴリーを開いたり閉じたりできます。ツールボックスはスクリプトにスクリプト要素をすばやく追加できるようにデザインされています。スクリプト・ペインにスクリプト要素を追加するには、ツールボックス・ウィンドウの要素名をダブルクリックするか、スクリプト・ペインにそれをドラッグします。

スクリプト・ナビゲーターは、スクリプトを含むプロジェクトにある要素(フレーム、ボタン、ムービークリップ・インスタンス)の階層リストを表示します。要素をクリックすると、スクリプト・ペインに、割り当てられているスクリプトが表示され、編集する目的にそってプロジェクト内のスクリプト全体をすばやく行き来できます。スクリプトが割り当てられた要素だけがスクリプト・ナビゲーターに現れます。

スクリプト・ペイン・ツールバーは、スクリプト・ペイン上部にあり、ボタンやコマンドを提供します。スクリプト・ペインに表示されている現在のスクリプトを、様々な方法で追加、編集できます。

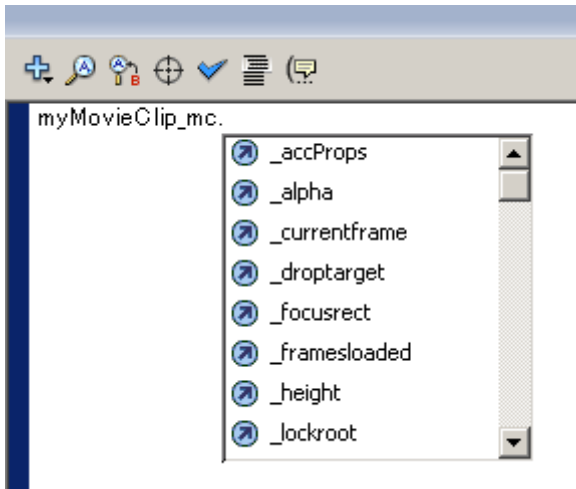
次のステップでは、ActionScript エディターの可能性と機能性を追及します。まずコード・ヒントから見ていきましょう。

4) スクリプト・ペインで、以下を入力します。

mymovieclip_mc.

NOTE ここではムービー内にこの名前のムービークリップ・インスタンスがあるものと仮定しています。

ドット(.)を入力するとすぐ、ドロップダウンメニューが出て、ムービークリップ・インスタンスに適用できるアクションのリストが表示されます。



なぜエディターはムービークリップ・インスタンス用のコマンドのリストを提供してくれるのでしょうか？それはムービークリップ・インスタンスにつけた名前によるのです。ムービークリップ・インスタンス名に `_mc` を加えたことに注目してください。この接尾辞によって、エディターは自動的に `myMovieClip_mc` がムービークリップ・インスタンスであるとみなし、スクリプト・ペインでその名前を入力すると、それに適切なコマンドのドロップダウン・リストを提供するのです。

一般的な、視認性のあるムービー要素用の接尾辞は、ボタンでは `_btn`、テキスト・フィールドには `_txt` となります。本書では、視認要素にはこれらの接尾辞を使用します。

他の、Sound や Data オブジェクトといった非視認要素にも接尾辞はあります。しかし非視認要素には接尾辞を使用する代わりに、Flash MX 2004 の新しい機能を活用します。それは次でお見せします。

NOTE 接尾辞の完全なリストは、ActionScript 辞書をご覧ください。

5) 現在のスクリプトを消去し、その場所に以下を入力します。

```
var mySound:Sound = new Sound();
```

このスクリプト行は、`mySound` という名前の新しい Sound オブジェクトを作成します。このシンタックスを使って Sound オブジェクトを作成すると、ActionScript エディターは `mySound` が Sound オブジェクトであると認識します。その結果、スクリプト内の後の部分でその名前がこのオブジェクトを参照すると、Sound オブジェクトに適切なドロップダウン・メニューが表示されます。ドロップダウン・メニューは関係するコマンドを自動的に

表示します。試してみましょう。

- 6) Enter または Return キーを押して、スクリプト・ペインの次の行に移動し、
以下を入力します。

```
mySound.
```

また再び、ドットを入力するとすぐに、Sound オブジェクトに適切なアクションのリストがドロップダウン・メニューで表示されます。

新しい Sound オブジェクトはこのシンタックスを使って作成できます。

```
mySound = new Sound();
```

しかし、このシンタックスでこのオブジェクトを記述しても、ステップ 5 でやったシンタックスのように、コードヒントの機能は有効になりません。

この設計の理解を助けるために、もう少し例をみていきましょう。

このコードは myColor という Color オブジェクトの、Color オブジェクトが関係するコードヒントを有効にします。

```
var myColor:Color = new Color();
```

このコードでは有効になりません。

```
myColor = new Color();
```

このコードは myDate という Date オブジェクトの、Date オブジェクトが関係するコードヒントを有効にします。

このコードは有効になりません。

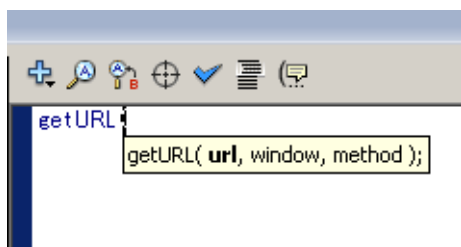
```
myDate = new Date();
```

NOTE 理解しやすくするため、本書では、ムービークリップ・インスタンスやテキスト・フィールド、ボタンなどの視認要素の名前をつけるときのみ、接尾辞を使用します。これは、視認要素は、このステップに記述したのと同じ方法では作成されないからです。それらの名前に接尾辞を使用することは、*ActionScript* エディターでそれらを参照する際、コードヒントが有効になる唯一の方法だからです。

次に、ActionScript エディターでコードヒントが有効になる他の場合を見てみましょう。

7) 現在のスクリプトを選択し消去します。その場所に以下を入力します。

GetURL(



開いた括弧を入力すると、コードヒント・ツールチップが現れます。この機能は、getURL() などの決まったパラメータを持つアクションのコードを入力すると、有効になります。現在決められるパラメータはツールチップの中で太文字で示されます。それぞれのパラメータ・データ(コンマで区切られています)を入力すると、ツールチップは更新されて、次の決められるパラメータを太文字で表示します。ツールチップは、アクションの閉じ括弧を入力すると消えます。

コードヒントは、(アクションの開いた括弧を入力したときのように)通常自動的にコードヒントが表示される場所にカーソルの挿入ポイントを置き、スクリプト・ペイン・ツールバーのコードヒント・ボタンを押すと、いつでも表示できます。



ツールバーの他のボタンでは、スクリプトの検索や語句の置換(名前を変更するときはずばらしく役に立ちます)、ターゲットパスの挿入やスクリプトのチェック(ムービーをまるごと書き出すという余計な時間を使うことなく)、自動フォーマット(エディターが、読みやすい形のスクリプトにする作業を受け持ちます)などができます。

ここで論じる ActionScript エディターの最後の特徴は、.as ファイルを作成する機能です。

8) ファイル・メニューから、新規を選択し、選択リストから ActionScript(AS) ファイルを選んでOKを押します。

このステップは新規 ActionScript ファイルを作成し、ActionScript エディターで編集する準

備を整えます。

ActionScript エディターは、前のステップで見たものと非常によく似ています。見た目でも分かりやすい違いは、エディターのインターフェイス要素が有効な間は、Flash のインターフェイスがぼやける点です。 .as ファイル編集中は Flash はこの “制限” モードに入ります。それはこのモードの唯一の目的が、.as ファイルを作成し編集することにあるからです。ドロー・ツールや他のパネル機能はこの場面では意味のないものです。

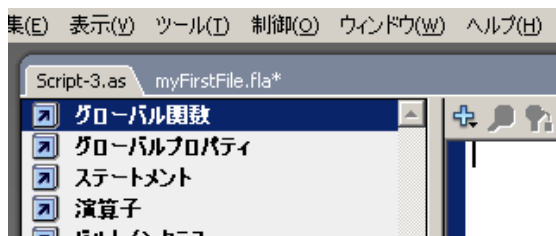
TIP F 4 キーを押して、ぼやけたインターフェイス要素を完全に消すこともできます。もう一度 F 4 キーを押せば元に戻ります。

もうひとつの分かりやすい違いは、スクリプト・ナビゲーターがないことです。これもこのモードが .as ファイルを編集するためのものであるからで、プロジェクトの様々なスクリプトをナビゲートするという機能は今では意味がありません。よってスクリプト・ナビゲーターはこのモードには存在しないのです。これらの違いを除けば、ActionScript エディターはこの練習で前に見たのと同じように動作します。

.as ファイルを作成したら（本書では後にやります）ファイル>名前をつけて保存、で保存します。

NOTE 本書ではほとんどの場合、特に指定する場合以外は、.as ファイルは（一度作成したら） main.flc ファイルと同じディレクトリに保存してください。

Flash オーサリング環境について気づく最後のことは、現在、ドキュメント・ウィンドウの上部に見える 2 つのタブです。ひとつのタブはステップ 1 で作成した Flash ドキュメントであり、もうひとつは今作った空の ActionScript ファイルです。これらのタブをクリックすると、オーサリング中の Flash ドキュメントと ActionScript ファイルを簡単にスイッチできます。タブをクリックすることで Flash は自動的に編集モードを変更します。



この練習を終えたら、次は最初のアプリケーション作りを始めます。

プロジェクトの計画

大量の ActionScript を含むプロジェクトを作成するときは、前もって計画だてるのが賢いやり方です。アイデアの段階で問題を処理することは、修正に時間がかかりフラストレーションもたまる開発段階で処理するよりもはるかに理にかなっています。長い目で見れば時間を節約することになるのは、我々が保証します。

何を生じさせたいのか？

これはスクリプトを計画する過程で最も大切な問いかけです。その答えの中で、できるだけクリアに、情報量を多く、視覚的に、しかしあまり細かなところまで突っ込むことは避けます。

このレッスンで論じるプロジェクトでは、電子支払いシステムのフロント・エンドとして機能するシーンを作成しようと思います。外部ソースからムービーにロードされる支払い額はテキスト・ファイルにしようと思います。ユーザーは、テキスト・ボックスに支払い額を入力できるようにしようと思います。ボタンが押されると、ユーザーの支払った額が、彼または彼女の所持金と比較され、払いすぎ、不足、残金ゼロという結果を視覚や文字で表します。ユーザーがボタンを離すと、シーンの視覚や文字要素はその元の状態に戻そうと思います。これを実行するスクリプトは、プロジェクトのメイン・スクリプトになります。

追跡する必要のあるデータは何か？

言い換えれば、そのアプリケーションで、なくてはならない数値や変数は何なのかということです。これから作るアプリケーションでは、そのデータとは電気代のことです。またユーザーの所有金額と使用金額の差を追跡する必要もあります。それによってカスタム・メッセージにその値を表示できるのです。

ムービー内でスクリプトが呼び起こされる前に何が必要なのか？

我々のプロジェクトでは、電子マネー額は、他のことがらが生じる前にムービー内で確立されていなければなりません。プロジェクトの第一のゴールは、電子マネー額と、支払うようにユーザーが選択した金額を比べることなので、もし電子マネー額がムービーの再生スタート時に確立されていないと、スクリプトが実行された時、比較するものがなくなってしまいます。スクリプトが実行される前に、またはムービーが再生を始める前に、データを作成し設定することはデータの初期化と言います。スクリプティングではよく行われることで、通常ユーザーに、この時点で、電子マネー額というデータがムービーにどういう方法で取り込まれるか考え始める必要があります。オーサリング時にムービー内にそれを配置することもできますし、ムービーの再生時に外部データ（例えばサーバーやテキスト・ファイル）からロードさせることもできます。このプロジェクトでは、後者を選びます。簡単なスクリプトを使ってムービー内に電子マネー額を持つテキスト・ファイルを口

ードします。データを提供するためにムービーにロードするテキスト・ファイルはデータ・ソースといいます。

どのイベントでメイン・スクリプトを引き起こすか？

これから作るプロジェクトでは、答えは明快で、それはボタンを押すことです。しかし Flash においてイベントはどのイベントでもスクリプトを始動させることができ、この疑問にいくつかの答えを準備することも重要なこととなります。ユーザーがマウスを動かし、押し、離れたとき、キーボードのキーを押したとき、何を起こす必要があるのか？ ムービークリップがシーンに始めて登場したときは？ イベントは継続的に起こさせる必要があるのか（ムービークリップ再生中ずっと）？ 次のレッスンではこういったイベントについて論じていきます。

メイン・スクリプトの始動後、決定しなければならない事項はあるのか？

ムービーのメイン・スクリプトが引き起こされると、ユーザーが支払うために入力した金額は、支払額が多すぎるか、少なすぎるか、ぴったりか決定するために、ユーザーの所持金と比較する必要があります。これらの問いの答えは、他の視認要素がスクリーンに見えるように、表示するカスタム・メッセージを決定します。

どの要素でシーンを作っていくか？ その機能をどうするか？

作成するシーンはいくつかの要素から形成されます。その中には名前をつけて ActionScript が使用し制御し、双方向的にやりとりできるようにします。スクリプトを始動させると、シーンは多くのサイトに見られる支払いボタンに見えるボタンが必要となります。

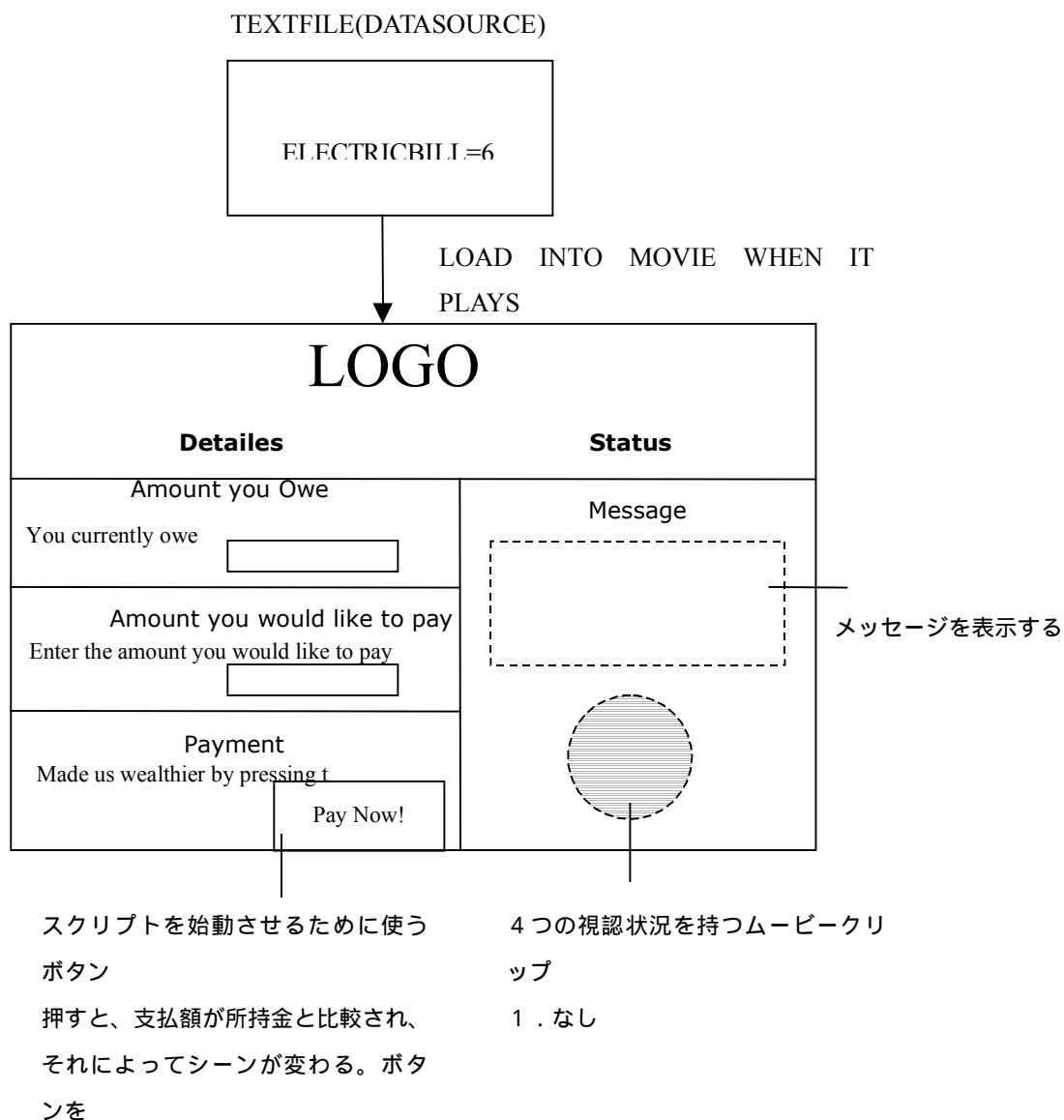
また所持金を表示するダイナミック・テキスト・フィールドも必要です。このテキスト・フィールドには `owed_txt` と名前をつけます。それに加えて、ユーザーが支払う金額を入力できるテキスト入力フィールドも必要です。このテキスト・フィールドには `paid_txt` という名前をつけます。またスクリプトによって生み出すカスタム・メッセージを表示するダイナミック・テキスト・フィールドも要ります。このテキスト・フィールドには `message_txt` という名前をつけます。最後に、ゴム印の形をした、4つの視認状態を表すムービークリップ・インスタンスを加えます。初めゴム印は見えません。ユーザーの支払額が不足すると、ゴム印が現れ、金額不足を表示します。ユーザーが正しい額を支払ったなら満額を表示します。支払いすぎの場合はそのように表示します。このゴム印のムービークリップ・インスタンスは、`stamp_mc` と名づけます。

シーンはどのように見えるか？

イラスト・ソフトでも紙と鉛筆でも何でもかまわないので、それを使って下に示す図によ

うな、(外見もそこに生じる動作も)シーンを現すラフスケッチを描きます。あつめた情報全てを書き込みます。計画段階において重要なこのパートはストーリーボーディングと呼ばれます。

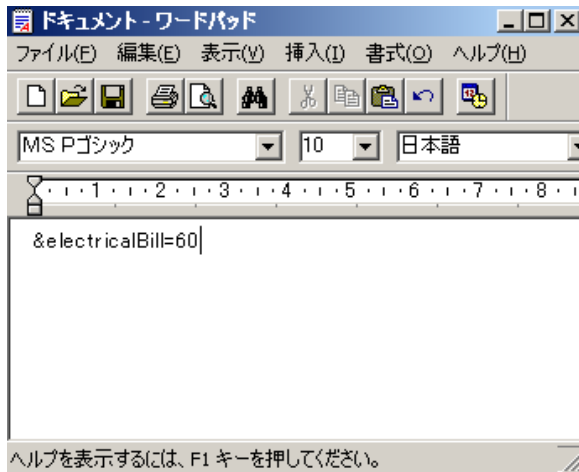
ActionScript が上達し、さらにプロジェクトを開発していくにつれて、以前の計画の問いを直感的にたずねる(そして答える)ことができるようになります。しかしスキルレベルには関係なく、ストーリーボーディングは計画段階の重要な部分です。



初めてのスクリプトを書く

では、プロジェクトのラフなストーリーボードなど、必要な情報が全部そろったところで、いよいよ作業を始めましょう。

- 1) Windows Notepad が、Macintosh SimpleText を開いて、新規テキストファイルを作成し、&electricBill=60 と入力します。



この練習の初めの部分では、ムービーの再生時にロードするデータのソースを作成します。我々のプロジェクトでは、データ・ソースは電子マネーを表す値を保持しています。Flashがこのテキスト・ファイルをロードすると、このテキスト行を解釈してムービー内にelectricBill（変数）という名前のデータのかたまりを作成し、60 という値を代入します。この変数の値はプロジェクトで何通りかの方法で使用されます。

NOTE 外部ソースから Flash にデータをロードすることは、Lesson 11 の“Flash からのデータの出し入れを行う”で詳しく論じます。変数は Lesson 6 “データの作成と操作”で詳しく論じます。

- 2) テキスト・ファイルに Electric_Bill.txt と名前をつけ、このレッスンのファイルのあるフォルダに保存します。Lesson01/Assets フォルダの electricbill1 を開きます。

スクリプトの設定を除いて、我々のプロジェクトでは大体同じ場所にプロジェクト要素は保存します。（本書の焦点は ActionScript であって、Flash の使い方ではありません）

- 3) プロパティ・インスペクタを開いて、“You currently owe:” の文字の右のテキスト・フィールドを選択します。プロパティ・インスペクタでこのテキスト・

フィールドの名前を `owed_txt` にします。



このテキスト・フィールドは所持金を表示するのに使用するので、`owed_txt` と名づけました。このテキスト・フィールド名に `_txt` という接尾辞を加えることによって、それを参照するスクリプトを作成すると、コードヒントが現れます。

- 4) **開いたままのプロパティ・インスペクタで、“ Enter the amount you would like to pay: ” の文字の右にあるテキスト・フィールドを選択します。プロパティ・インスペクタでこのテキスト・フィールドに `paid_txt` のインスタンス名を与えます。**

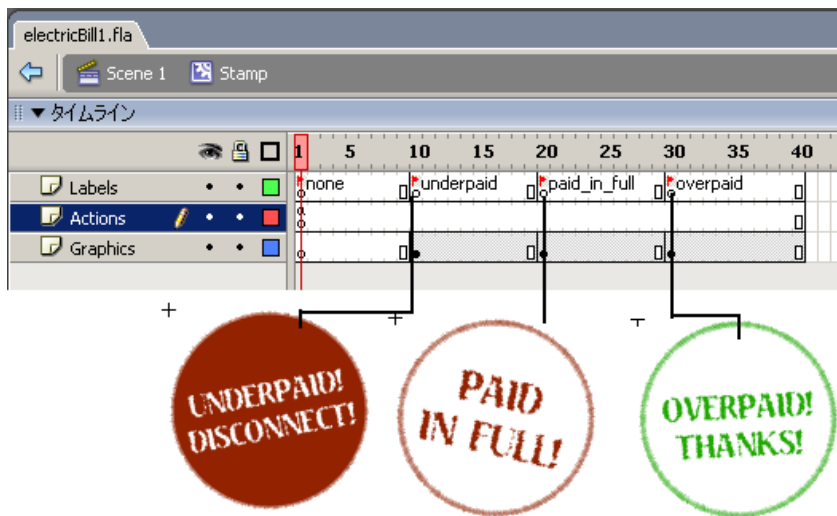
選択したこのテキスト・フィールドはテキスト入力です。これはユーザーの入力を受け付けるために使用します。この場合は、ユーザー支払いを希望する額を入力します。このテキスト・フィールドには `0` を入れているので、ムービーが再生されると、初めからその額 `0` が表示されます。

- 5) **ステージの右側、“Status”の文字の下にあるテキスト・フィールドを選択します。プロパティ・インスペクタで、このテキスト・フィールドに `message_txt` というインスタンス名をつけます。**

このテキスト・フィールドはユーザーに対してカスタム・メッセージを表示するために使用します。これはユーザーが支払うと決めた額によって変化します。

- 6) **ムービークリップ・インスタンス（これは `message_txt` テキスト・フィールドの下にある、小さな白い円です）を選択し、`stamp_mc` と名づけます。**

我々のスクリプトはこのムービークリップ・インスタンスに影響を及ぼすので、このムービークリップ・インスタンスに名前をつけなくてはなりません。そうすることでそのムービークリップ・インスタンスに対する処理をスクリプトに伝えることができるようになります。このムービークリップ・インスタンスは、そのタイムライン上に4つのフレーム・ラベル (`none, underpaid, pai_in_full, overpaid`) を持ち、条件によって見え方を変えます。初めは `none` の状態が見えています。



今書こうとしているスクリプトでは、ユーザーが支払うべき額よりも少なく支払った場合は、UnderPaid 印を表示させるように、ムービークリップを *underpaid* ラベルに移動させます。支払うべき金額ちょうどを支払った場合は、Paid in Full 印を表示させるように、ムービークリップのタイムラインを *paid_in_full* ラベルに送ります。ユーザーが支払いすぎた場合には、Overpaid 印を表示させるように、クリップのタイムラインをフレーム・ラベル *overpaid* に移動させます。

シーンの要素に名前をつけ、スクリプトを書く準備はできました。まずはムービーに外部テキスト・ファイルからデータをロードする命令から始めます。

7) アクション・パネルを開け、アクション・レイヤーのフレーム 1 を選択し、下記スクリプトを入力します。

```
var externalData:LoadVars = new LoadVars();
```

アクションスクリプトは、LoadVars オブジェクトを使って外部ソースから来るデータをロードしたり保持します。サンプルの LoadVars オブジェクトは、前に作成した *Electric_Bill.txt* という外部テキスト・ファイルです。

次のステップでスクリプトを書き加え、外部テキスト・ファイルにあるデータをこのオブジェクト内にロードします。

8) フレーム 1 の現在のスクリプトの下に以下を加えます。

```
externalData.onLoad = function(){
    owed_txt.text = externalData.electricBill;
}
```

この3行のスクリプトでは、externalData という LoadVars オブジェクトにテキストファイルからのデータをロードし終わったら何をするかということ、Flash に伝えてあります。残りのスクリプトを実行させるきっかけとして onLoad イベントを使用しています。このスクリプトは主として、“データが externalData LoadVars オブジェクトにロードされたら (onLoad)、以下のファンクションを実行せよ” (ファンクションについては Lesson 5 “ファンクションを使う” でさらに学びます)と言っています。

このファンクションは1つのことを行います。owed_txt テキスト・フィールド・インスタンスの text プロパティ値を、externalData.electricBill の値に等しくさせることです。これには少々説明が要ります。

使用している LoadVars オブジェクトは externalData という名前をつけました。前に説明したように、このオブジェクトは外部ソースからのデータをロードしたり保持するために使用されます。テキスト・ファイルには、60 という値を持っている electricBill という名前の1つのデータが含まれています。このデータが externalData LoadVars オブジェクトにロードされると、そのオブジェクトの一部になります。そして、外部テキスト・ファイルのデータのロードが終了するとすぐ、externalData.electricBill は 60 という値を保持します。もし外部テキスト・ファイルが他のデータを持っている場合は、以下のシンタックスで参照できます。

externalData.name

externalData.address

externalData.phone

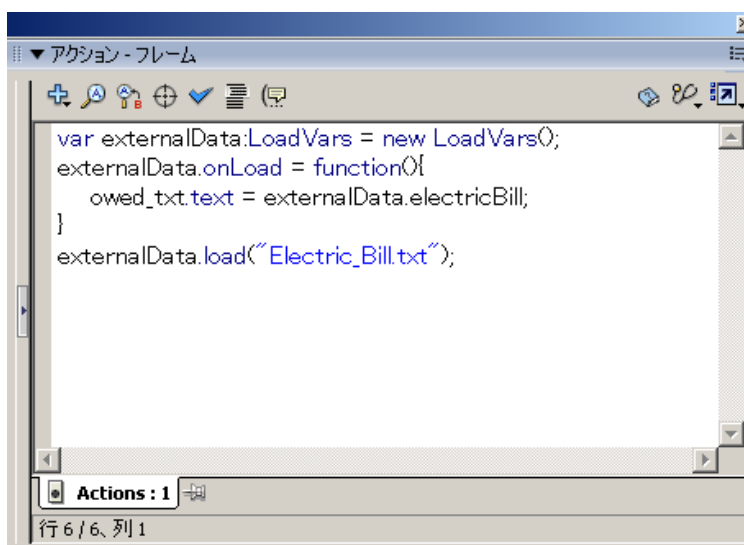
owed_txt は顧客が支払うべき額を表示するために使われるテキスト・フィールドに割り当てたインスタンス名ということをお出ししてください。このようにして、このファンクションは 60 という数字を owed_txt テキスト・フィールドに表示させるのです。

テキスト・フィールドは非常に多くのプロパティを持つオブジェクトです。テキスト・フィールド・インスタンスのプロパティの1つはその text プロパティです。このプロパティは、そのインスタンス内に表示するテキストを指定するために使用されます。owed_txt と名づけたテキスト・フィールド・インスタンスにテキストを表示するよう設定するために、どのようにこのプロパティが使われるか、スクリプトがはっきり示しています。オブジェクト名とそのプロパティ名がドット(.)で分けられていることに注意してください。ActionScript では、このようにしてオブジェクトに関係するものを機能するよう指定します。この場合はそのプロパティになります。

またここでは、オブジェクトのプロパティとその変数（とその値）という2つの要素の間にある演算子の使用にも注意しましょう。ここでは等記号(=)は、externalData.electricBill 変数の値に owed_txt テキスト・フィールド・インスタンスの text プロパティを代入するようスクリプトに伝えるために使用されています。

9) フレーム1の現在のスクリプトの下に以下を加えます。

```
externalData.load("Electric_Bill.txt")
```



このスクリプト行は、Flash に外部データのロードを開始するよう伝えます。これは、LoadVars オブジェクトが使用する特別なコマンド、load コマンドを使って行います。このコマンドには、ロードされる外部データの URL を指定することが必要です。今の場合、Electric_Bill.txt という名前のテキスト・フィールドです。これはパラメータ値の設定例です。

前に、アクションはどのようにしてパラメータをとるか、そして与えたそのパラメータ値が、アクションがどのように動作するかを決めるということを論じたことを思い出してください。今の場合、load アクションによって、ハード・ドライブ上の、Web 上のテキストベースのデータ・ソースを示す URL を入力できます。コマンドは常に同じやり方で動作します(外部ソースからデータをロードする)が、データ・ソースのありか(パラメータ値)は変更できます。ActionScript の作業をすすめる中で、このやり方をするアクションは多いことに気づかれることでしょう。

スクリプトが要求した、外部テキスト・ファイルからデータを Flash へロードすれば終了で

す。

手短かに復習してみましょう。

初めに、外部データをロードし保持する `externalData` という `LoadVars` オブジェクトを作りました。次に、`externalData` オブジェクトに対し、データのロードが終了した後のアクションを実行するよう伝えました。最後に、外部データのロード処理を開始しました。

データをロードする前になぜロードされたデータをどう処理するかというスクリプトを書くのか、不思議に思われているかも知れませんが、すぐには分からないかも知れませんが、そうすることが論理的なのです。自然界を見てみましょう。例えば、我々の胃は食物を消化するよう予めプログラムされています。もし胃が食物をどう処理するか知る前に、食事をとらなければならないとしたら、その後起こるであろう大変な事態を想像してみてください。同じことがこれにも当てはまります。スクリプティングでは一般的に、現実はその結果が生じる前に、様々な結果を制御するスクリプトを書いておく必要があります。

NOTE このスクリプトはムービーのフレーム 1 に配置していますが、そのことでムービーが再生されるといち早くこのスクリプトは実行されます。このことは、`Pay Now!` ボタンが機能するようにこれから追加するスクリプトより先に、ムービーが `electricBill` の値を必要とするので、重要なことです。

では `Pay Now!` ボタンのスクリプトを書いていきましょう。

10) アクションパネルを開いたままで、`Pay Now!` ボタンを選択し、以下のスクリプトを入力します。

```
on(press){
    var amountPaid:Number = Number(paid_txt.text);
    var amountOwed:Number = Number(owed_txt.text);
}
```

このスクリプトは、これがアタッチされたボタンが押されたとき実行されます。スクリプト内では、このイベントに開いた中括弧(`{`)がつづき、2行のスクリプト行、そして閉じ中括弧(`}`)で終わります。中括弧は、“ボタンが押されたとき、これら2つのkとをせよ”ということを示しています。

スクリプトの1行めでは、`amountPaid` という名前の変数が作成され、その値が、`paid_txt` テキスト・フィールド・インスタンスに表示されたものと等しくなるよう設定されています。通常、テキスト・フィールドに入力されたものはテキストであると考えられます。よ

って、たとえ 100 の値が数字としてテキスト・フィールドに表示されていても、それは数値の 100(クォテーションマークなし)ではなく、テキスト (1,0,0,または”100”の文字からなる) テキストとみなされます。

NOTE 通常 *Flash* はテキスト・フィールドに入力されたものは何でもテキストであるとみなします。たとえユーザーがテキスト・フィールドに入力するときクォテーション・マークを加えなくても、*Flash* はクォテーション・マークを追加するので、ムービーはその値がテキスト値であると理解します。

ActionScript が数値として認識するようになる、テキスト値を数値に変換できる特別なツールの `Number()` ファンクションについて考えます。ファンクションの括弧の中に変換したいテキストを入れます。例えば、

```
var myNumber:Number = Number(“945”);
```

は、テキスト”945”を数値の 945(クォテーション・マークなし)に変換し、`myNumber` という名前の変数にその数値を代入します。我々のスクリプトでは、`Number()` ファンクションを使用し、ファンクションの括弧内に変換されるテキスト値への参照を配置しました。ユーザーが `paid_txt` フィールドに”54”(初めはテキスト)を入力すると、`Number()` ファンクションが 54 の数値を `amountPaid` に与えます。

TIP `Number()` ファンクションには制限があります。数値になる可能性のあるものを変換することにのみ使用できます。例えば `Number(“dog”)` は、数値への変換が不可能なので、結果として `Nan`(数値ではない)になります。

ステップ 10 の次のスクリプト行は、基本的におなじことをやっいて、ただ `amountOwed` の値を `owed_txt` フィールド・インスタンスに表示される数値に変換される値に等しくしているだけです。このようにして、変換が行われた後、`amountOwed` には 60 の値が与えられます。

このようにテキスト・フィールドの値を変換するのは、スクリプトのほとんどがその値を数学的な評価や計算に利用するためです。動作させるためには、テキストでなく数値として値を認識する必要があります。

要約すると、ボタンが押されると、`paid_txt` と `owed_txt` テキスト・フィールドに表示されるテキスト値は数値に変換されます。それらの数値は各々 `amountPaid` と `amountOwed` という名前の変数に代入されます。これらの変数値はスクリプトの残りの部分で使用されます。

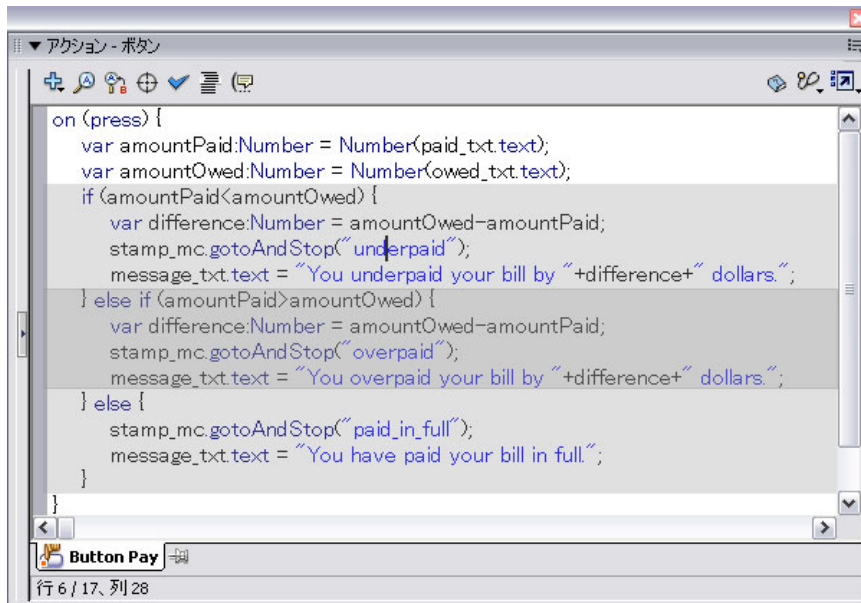
11) アクション・パネルを開けたままで、前のステップで作成したスクリプトに以下の行を加えます。スクリプトは中括弧の間に追加します。var amountOwed:Number = Number(owed_txt.text);の下です。

```
if(amountPaid < amountOwed) {
    var difference:Number = amountOwed - amountPaid;
    stamp_mc.gotoAndStop("underpaid");
    message_txt.text = "You underpaid your bill " + difference + " dollars.";
} else if(amountPaid > amountOwed) {
    var difference:Number = amountOwed - amountPaid;
    stamp_mc.gotoAndStop("overpaid");
    message_txt.text = "You overpaid your bill " + difference + " dollars.";
} else {
    stamp_mc.gotoAndStop("paid_in_full");
    message_txt.text = "You have paid your bill in full";
}
```

on(press)イベントの中括弧の間にこれらのスクリプト行を加えたので、ボタンが押されるとこのスクリプトも実行されます。

スクリプトのこの部分は、以下の行で3つに分けられています。

```
if(amountPaid < amountOwed)
if(amountPaid > amountOwed)
else
```



これら3つの行は、スクリプトが実行されたとき、スクリプトが分析する一連の条件を表しています。それぞれの場合で分析される条件は括弧の間で特定します。これら各行の下には、インデントされた何行かのスクリプトがありますが（追加した中括弧の間）、それらは特定の条件が真であった場合実行されるアクションです。このように動作します。

Play Now! ボタンが押されると、ユーザーが支払うよう入力した金額が、その支払うべき額より多いのか、少ないのか、等しいのかをスクリプトで決定します。それはスクリプトが分析する3つの条件で全て行います。最初のファンクションを見てみましょう。それはこのようになっています。

```
if(amountPaid > amountOwed){
    var difference:Number = amountOwed - amountPaid;
    stamp_mc.gotoAndStop("underpaid");
    message_txt.text = "You underpaid your bill by " + difference + " dollars.";
}
```

1行目は、比較演算子(<)を使って、ある変数値と他の値を比べています。それは基本的に、もしユーザーが入力した支払額 (amountPaid) が、そのユーザーの支払うべき金額 (amountOwed) より少なかったら、以下のアクションを実行する “ という意味です。これらのアクションはその条件が true のときのみ実行されます。そしてそれらが実行されるときは、グループとして実行されます。それは、中括弧内の各セットの中に配置されているからです。最初のアクションは difference という名前の変数値を作成し、amountOwed から amountPaid を引いた値をその変数に代入しています。もしユーザーが支払額として 40 を入力したら、difference は 20 になります (60 - 40)。左側に値を代入するより先に、等号の右

側が計算されると知っておくことは重要です。次の行は、`stamp_mc` ムービークリップ・インスタンスにフレーム・ラベル `underpaid` に行って止まるように伝えています。これによって `Underpaid` 印が表示されます。最後の行は `message_txt` テキスト・フィールドに表示されるカスタム・メッセージを作成しています。

我々は、メッセージがスクリプト内で作られるという理由からこれをカスタム・メッセージと呼んでいます。この行の `difference` 変数の使用に注目してください。Difference の値がこのメッセージの中ほどのクォテーション・マークのついたテキストとプラス記号の間に挿入されています。もし `difference` が 20 なら、このメッセージはこう読めます、“ You underpaid your bill by 20 dollars ”。

クォテーション・マークではさまれたものは何でもただのテキストとみなされることを思い出してください。difference はクォテーション・マークではさまれていないので、ActionScript は、それが変数名を参照しているということを知っていて、そこにその変数値を挿入するのです。プラス記号(+)は、連結して1つのメッセージを作成するために使用されます。等号(=)は、`message_txt` テキスト・フィールドの `text` プロパティに、連結された値を代入します。

ユーザーが入力した支払額が、支払うべき金額と同じか、多い場合、スクリプトのこの部分は無視され、次の部分が分析されます。次はこうです。

```
}else if(amountPaid > amountOwed) {  
    var difference:Number = amountOwed - amountPaid;  
    stamp_mc.gotoAndStop("overpaid");  
    message_txt.text = "You overpaid your bill " + difference + " dollars.";  
}
```

1行目はこのように言っています。“もしユーザーの支払額が、支払うべき金額より多いなら、これらのアクションを実行せよ”。ここで実行されるアクションは、前に論じたことをマイナス差に応用したものです。1つめの違いは、`stamp_mc` ムービー・クリップ・インスタンスがラベルのついたフレーム `overpaid` に送られることです。それによって `Overpaid` 印が表示されます。2つめの違いは、カスタム・メッセージの語句です。`underpaid` という代わりにここでは `overpaid` と言っています。このセクションのアクションは、ユーザーが支払うべき金額より多く支払った場合のみ実行されます。それにあてはまらない場合は、これらのアクションは無視され、最後の部分が分析されます。その部分はこうです。

```
else{  
    stamp_mc.gotoAndStop("paid_in_full");
```

```
message_txt.text = "You have paid your bill in full";  
}
```

ここでは、スクリプトは amountPaid が amountOwed より多いか少ないかを聞くようには始まっていません(前の2つのセクションではそうだったのですが)。これは、ユーザーが支払うべき正確な額を入力した場合のみここまでスクリプトが実行されるからです。スクリプトのこの部分は最初の2つのセクション両方が実行されない場合のみ実効されます。

最後に、ボタンが押されると、これら3つのアクション・セットの1つだけが実行されます。その結果、stamp_mc ムービークリップ・インスタンスが一定の方法で表示され、カスタム・メッセージが message_txt テキスト・フィールドに表示されます。

Pay Now!ボタンが離されると、シーンの要素は全部リセットされ、ムービーが最初に再生されたときのように戻そうと思います。では以下の機能を追加しましょう。

12) アクション・パネルを開いたまま、Pay Now!ボタンを選択し、現在のスクリプトの最後に以下の行を追加します。

```
on(release){  
    stamp_mc.gotoAndStop("none");  
    messege_txt.text = "";  
}
```

```
on (press) {
    var amountPaid:Number = Number (paid_txt.text);
    var amountOwed:Number = Number (owed_txt.text);
    if (amountPaid < amountOwed) {
        var difference:Number = amountOwed - amountPaid;
        stamp_mc.gotoAndStop ("underpaid");
        message_txt.text = "You underpaid your bill by " + difference + " dollars.";
    } else if (amountPaid > amountOwed) {
        var difference:Number = amountOwed - amountPaid;
        stamp_mc.gotoAndStop ("overpaid");
        message_txt.text = "You overpaid your bill by " + difference + " dollars.";
    } else {
        stamp_mc.gotoAndStop ("paid_in_full");
        message_txt.text = "You have paid your bill in full.";
    }
}

on (release) {
    stamp_mc.gotoAndStop ("none");
    message_txt.text = "";
}
```

このスクリプトはボタンが離されると引き起こされます。stamp_mc ムービークリップ・インスタンスをフレーム・ラベル none に送り、message_txt テキスト・フィールドを空にして、これらのシーン要素を元の状態に戻します。

13) このファイルを electricBill2.fla の名前で、前に作った electricBill.fla を保存した同じディレクトリに保存します。

このファイルはこのレッスンの次の練習で使用します。

初めてのスクリプトのテスト

ActionScript が計画通りにいつも動作するか考えるとわくわくするものです。手紙を書くとときコマを忘れたり綴りを間違ったりするように、スクリプトを書くときも、ActionScript への習熟度に関わらず、簡単に間違いを犯してしまいます。しかし手紙の受取人の場合と違って、Flash はスクリプト・エラーを許しません。スクリプティングでは、エラーはバグであり、バグは、スクリプトがまったく動作しないか、計画通りに動作しないかどちらかを意味します。ありがたいことに、Flash ではいくつかの簡単な方法でスクリプトをテストしバグをつぶす方法があります。

1) electricBill2.fla が開いていなければそれを開きます。

これは先の練習で作成したファイルです。このレッスンでは、Flash 内部のテスト環境でプロジェクトの機能性をテストします。

2) Flash のメニュー・バーから、制御>ムービー・プレビューを選択します。

このコマンドは、Flash のテスト環境内で、全部の機能を持つムービーとして書き出し、表示します。この環境内では、(全ファイル・サイズやストリーミング設定、外見を決定するなど)ムービーをテストできる全ての方法がそなわっているのですが、その双方向的部分のテストに関心は注がれます。それは試せるところは全部試す、ということです。

TIP できるだけ多くの友人や同僚に参加してもらい、プロジェクトのテストを助けてもらいます。これは、あり得る場合をテストする大きな機会であり、結果潜在的なバグを見つけ出せるのです。

3) 様々な金額を”Enter the amount you would like to pay:”内のテキスト・フィールドに入力し、Pay Now!ボタンを押します。

- ・ 60 以下の金額を入力します。このテキスト・フィールドに 35 を入力すると、メッセージ “ You underpaid your bill by 25 dollars. ” が表示され、`stamp_mc` ムービークリップ・インスタンスが Underpaid 印を表示するはずですが。
- ・ 60 以上の金額を入力します。このテキスト・フィールドに 98 を入力すると、メッセージ “ You have overpaid your bill by 38 dollars. ” が表示され、`stamp_mc` ムービークリップ・インスタンスが Overpaid 印を表示するはずですが。これが正しい動作なのですが、実際はメッセージは、38dollars ではなく overpayment of -38dollars と表示されます。このバグをエラーと記録しテストをつづけます。
- ・ ぴったりの金額 60 を入力します。このテキスト・フィールドに 60 を入力すると、メッセージは “ You have paid your bill in full ” と表示され、`stamp_mc` ムービークリップ・インスタンスは Paid in Full 印を表示するはずですが。
- ・ このフィールドの文字を消去します。文字を消して Pay Now!ボタンを押すと、メッセージは “ You have paid your bill in full ” となり、`stamp_mc` ムービークリップ・インスタンスはぴったりの額を支払ったことを表示します。明らかにこれはまずいです。これをエラーと記録しテストをつづけます。
- ・ テキストを入力します。アルファベットで始まる文字を入力して Pay Now!ボタンを押すと、メッセージは “ You have paid your bill in full ” となり、`stamp_mc` ムービークリップ・インスタンスは Paid in Full 印を表示します。これも明らかなミスでエラーです。

TIP 複雑なプロジェクトをテストしているときにバグを発見したら、テストをやめすぐにバグつぶしを始めることが一番よいときがあります(いくつかのバグを記録し、その後一度にその修正を行うのとはまったく逆です)。その理由は、バグをなくそうとすると、うっかり新たなバグを作り出してしまう恐れがあるからです。一度にいくつかのバグを修正することは、結果として、明らかに望みもしない新たなバグを作り出してしまうのです。

1 回に1つのバグを修正することで、バグつぶしの集中も高まりますし、必要のない多くの後戻りした作業も避けることができます。

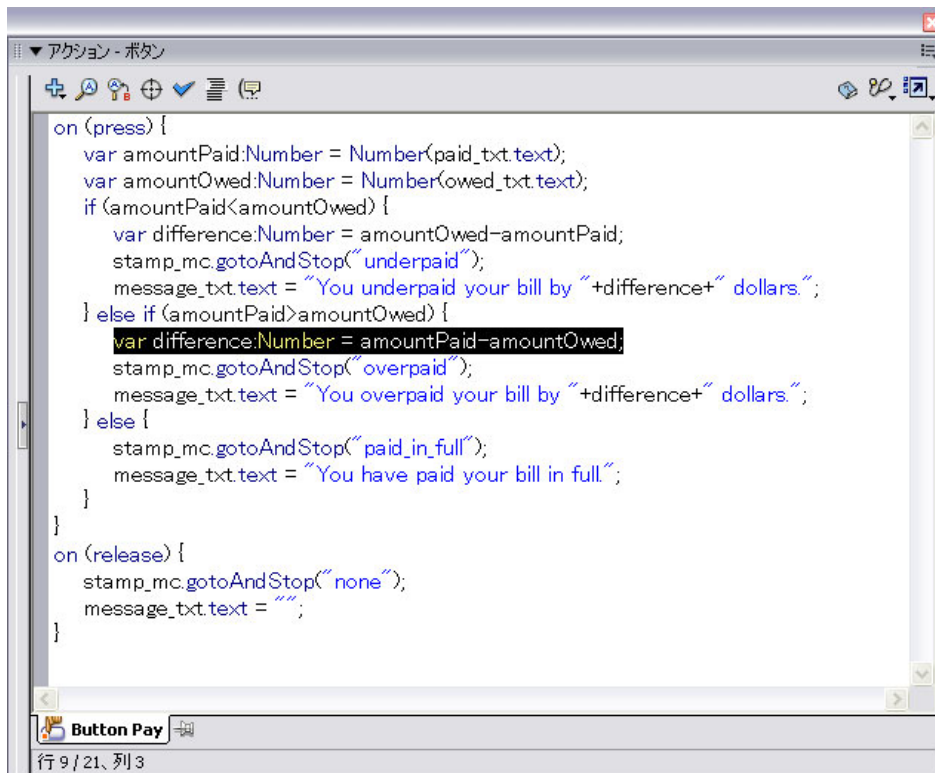
ご存知のように我々のプロジェクトは3つのバグを含んでいます。

- ・ ユーザーが多く支払うと、その過剰分がカスタム・メッセージに負の数値として表示されます。
- ・ ユーザーが何も支払わないことを選択すると、プロジェクトは正しく機能しません。
- ・ ユーザーが数値でなくテキストを入力すると、プロジェクトは正しく機能しません。

なぜこれらのバグが発生するのか考えてみましょう。1つめのバグの場合、このダイナミック・メッセージに表示される数値は、スクリプトが実行されたとき `difference` 変数が作り出すその値に基づくことが分かっています。さらに、ユーザーが支払うべき額より多く支払うときのみこの問題が起こることも分かっています。よって問題は、ユーザーが多く支払ったとき `difference` が計算される方法にあるのです。スクリプトのその部分をみてみましょう。

他の2つのバグについては、スクリプトは、それが実行されたとき、ユーザーがいくらの支払額を入力したかによって、様々な方法で動作するよう書かれています。しかしユーザーが何も入力しないかもしれない、テキストを入力するかもしれない、その両方がプロジェクトをおかしくしてしまうかもしれない、ということを忘れていました。スクリプトに少し修正を加えましょう。

- 4) **テスト環境を閉じ、オーサリング環境に戻ります。Pay Now!ボタンを選択し、9行目のスクリプトを修正します。それは現在こうなっています。 `var difference:Number=amountOwed - amountPaid` これをこのように直します。 `var difference:Number=amountPaid - amountOwed`。**



```
on (press) {
    var amountPaid:Number = Number(paid_txt.text);
    var amountOwed:Number = Number(owed_txt.text);
    if (amountPaid < amountOwed) {
        var difference:Number = amountOwed - amountPaid;
        stamp_mc.gotoAndStop("underpaid");
        message_txt.text = "You underpaid your bill by "+difference+" dollars.";
    } else if (amountPaid > amountOwed) {
        var difference:Number = amountPaid - amountOwed;
        stamp_mc.gotoAndStop("overpaid");
        message_txt.text = "You overpaid your bill by "+difference+" dollars.";
    } else {
        stamp_mc.gotoAndStop("paid_in_full");
        message_txt.text = "You have paid your bill in full.";
    }
}

on (release) {
    stamp_mc.gotoAndStop("none");
    message_txt.text = "";
}
```

amountPaid が amountOwed より大きくなったときどんな動作を起こすかを決定するスクリプトの部分を見直してみると、difference は amountOwed から amountPaid を引いて計算されていることが分かります。これのどこが問題なのでしょう？ ユーザーが 84 ドル払うと計算される差は 60 - 84 (amountOwed - amountPaid) です。小さな数値から大きな数値を引くと結果は負の数値になります。この問題を修正するには、単純に difference の値を設定するスクリプト行の amountOwed と amountPaid の位置を入れ替えます。すると、小さな数値は大きな数値から引かれ、結果は正の数値になります。

NOTE difference の値が設定されている、スクリプトの他の部分は修正する必要はありません。なぜなら、その部分はユーザーが支払うべき額より少なく支払ったときにだけ実行される部分で、この場合、値は正しく計算されるからです。

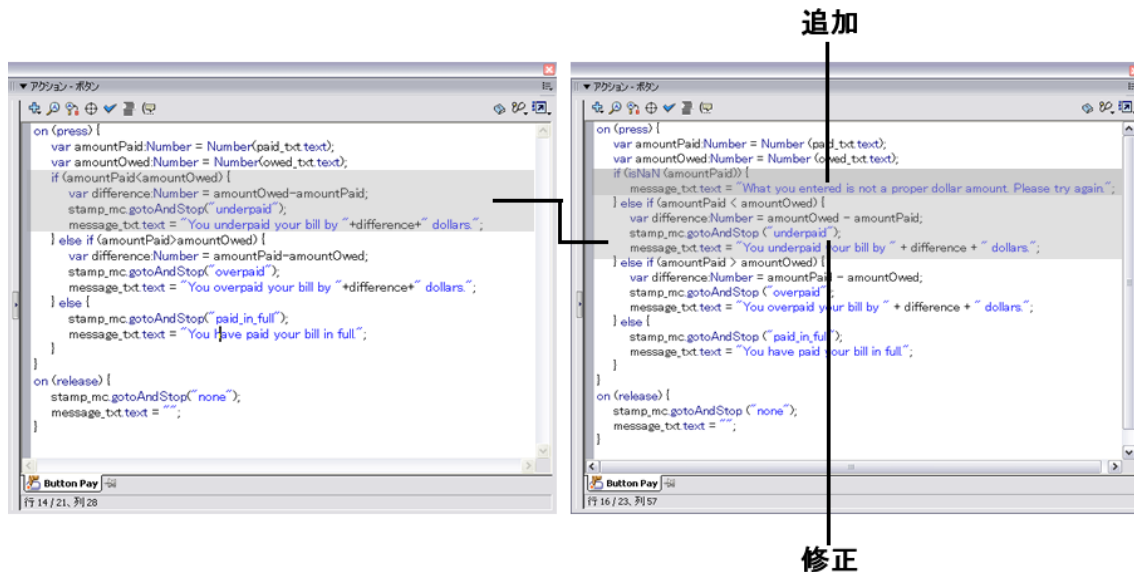
5) Pay Now! ボタンを選択しアクション・パネルを開いたままで、if ステートメントに追加と修正をします。

追加：

```
if(isNan (amountPaid)){
    message_txt.text = "What you entered is not a proper dolalr amount. Please try again.";
}
```

修正 ;

```
} else if (amountPaid < amountOwed) {  
    var difference:Number = amountOwed - amountPaid;  
    stamp_mc.gotoAndStop("underpaid");  
    message_txt.text = "You underpaid your bill by " + difference + " dollars.";  
}  
}
```



この追加によって、ユーザーが支払うべき金額として何も入力しなかったり、テキストを入力するなどした、思いもよらないことに対処できるようになります。Pay Now! ボタンが押されると、amountPaid の値が数値でない(isNaN)なら何もしないが、ユーザーが適切なドルの金額を入力したら、それにこたえるメッセージを表示しろ、とスクリプトは言っています。

もし入力された金額("frog"など)が数値に変換できない場合や、フィールドが空白のままだったときには、スクリプトのこの部分が実行されます。isNaN()ファンクションは、ActionScript が単純だが重要なタスクを処理するために提供する特別なツールです。このファンクションの括弧の中に、文字の値("cat"とか 57)を入れる代わりに、変数への参照(今の場合は、amountPaid)を入れていることに注意してください。これによって、変数の保持する値が分析されます。

NOTE isNaN()ファンクションとその使用法は本書を通して非常に細かなところまで扱います。

この部分はスクリプトが最初に行われるとチェックされる最も論理的な部分なので、最初の条件文にもって来ました。ユーザーが数値を入力すると、スクリプトのこの部分は無

視され、スクリプトの残りの部分が考えていたとおりに実行されます。

- 6) Flash のメニュー・バーから**制御>ムービー・プレビュー**を選択します。書き出されたテスト・ファイルで、“Enter the amount you would like to pay”のテキスト・フィールドにいろいろな値を入れ、Pay Now!ボタンを押します。

この時点で、ムービーはあらゆる状況で正しく動作するはずですが。

- 7) **テスト環境を閉じ、オーサリング環境に戻します。このファイルを electricBill3.fla として保存します。**

おめでとうございます！最初のレッスンが終了しました。

学んだこと

このレッスンでは、以下のことを学びました。

- ・ ActionScript2.0 の紹介を受け、2.0 と 1.0 との相違点、似ている点について学びました。
- ・ スクリプトを構成するさまざまな要素に馴染みました。
- ・ ActionScript エディターの使用法を学びました。
- ・ ActionScript プロジェクトの計画と開発について学びました。
- ・ プロジェクト・スクリプトを書きました。
- ・ スクリプトをテストし、バグを探しました。
- ・ バグを修正し、プロジェクトを完成させました。